December 1993

Report No. STAN-CS-TR-93-1498

Also numbered KSL-93-69

PB96-149109

Diagnosis using Action-Based Hierarchies for Optimal Real-Time Performance

by

David Ash

Department of Computer Science

Stanford University

Stanford, California 94305

DIAGNOSIS USING ACTION-BASED HIERARCHIES FOR OPTIMAL REAL-TIME PERFORMANCE

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By
David Ash
December 1993

Abstract

To respond effectively in real time, an agent needs to be able to deal effectively with differing availability of information. In particular, although the agent ideally would like to be able to diagnose one or more particular faults that may be present at a given time, a deadline may prevent it from doing so because prior to the deadline it is required to act or face a catastrophic outcome. Thus, an agent should have available to it a set of actions some of which are appropriate, but not optimal, for large classes of faults and some of which are appropriate for specific faults.

Tests are available to help the agent diagnose a fault; given this framework decision trees would provide one possible approach to this diagnosis problem. However this work shows that the information-theoretic heuristic commonly used to structure decision trees is not ideal in a real time domain because it concerns itself with reaching a leaf node as fast as possible, not with the value of the actions it might obtain along the way. My thesis presents an alternative heuristic, the action-based heuristic, which may be used to structure a decision tree; the resulting structure together with the actions themselves is called an action-based hierarchy.

The approach is validated in several ways. First a complexity analysis is undertaken to show that the complexity of the structuring algorithm is not prohibitive. Then some theoretical results are given; this is followed by experiments both with abstract inputs and inputs from a real-time domain, surgical intensive care unit patient monitoring. The thesis concludes with a description of an implementation of these ideas in a system known as ReAct.

Acknowledgements

The successful completion of this thesis would not have been possible were it not for my adviser, Barbara Hayes-Roth. It was Barbara who first suggested to me the idea of working on the problem of providing response to critical events in the face of insufficient diagnostic resources. Barbara has helped me develop my ideas on this problem from vague beginnings to a finished thesis. During my many discussions with her about my work, she has always been able to immediately pinpoint the flaw or weak spot in my thinking and therefore has always encouraged me to improve the quality of my work. Barbara has also provided the many resources--from computers to contacts in the medical community--that have been essential to the completion of this work.

I thank Nils Nilsson for serving on my thesis committee and acting as my co-adviser. Nils has provided many good pointers to related areas of research that have proven to be invaluable in sharpening the ideas in this thesis. In particular, the idea of comparing the value of action-based hierarchies to that of classical decision trees grew out of a question Nils asked at one of our committee meetings.

Thanks are also due to David Gaba for serving on my thesis committee. As the representative of the medical community on my committee, David has helped to ensure that the problem I attacked, as well as the solution I proposed, was relevant in the medical domain of application. David also provided valuable assistance in verifying the accuracy of the medical knowledge base used as the basis for the domain experiments in this thesis.

Many past and present members of the Adaptive Intelligent Systems (AIS) group provided valuable feedback during weekly group seminars. In no particular order, they include Rich Washington, Janet Murdock, Vlad Dabija, Serdar Uckun, Anne Collinot, Angel Vina, Michael Wolverton, Lee Brownston, Philippe Morignot, and Adnan Darwiche.

This thesis owes an intellectual debt to the work done on a predecessor system developed by Adnan Darwiche for his MSAI project in the Knowledge Systems Laboratory. I have also enjoyed the opportunity to discuss my work on several occasions with Adnan, and have always benefitted from the discussions.

My medical colleagues have provided considerable assistance to me in developing a plausible medical knowledge base. In particular, Garry Gold is the primary author of the medical knowledge base upon which my experiments are based. Both David Gaba and Adam Seiver have helped to ensure the medical validity of the knowledge base. Early discussions with medical colleagues Ida Sim and Michael Falk helped to formalize the medical problem.

Helpful comments on this work have been given by a number of people at different times and for different aspects of the work including Thomas Dean, Andrew Goldberg, Brad Efron, Victor Lesser, Yuval Shahar, and the anonymous reviewers of various conference and journal articles.

Financial support has been provided by DARPA through NASA grant NAG2-581 (ARPA order 8607) and the DSSA project (DARPA contract DAAA21-92-C-0028). Thanks are also due to Edward Feigenbaum for sponsoring the work at the Knowledge Systems Laboratory.

My friend Ron has been a bastion of support throughout the long process of obtaining my Ph.D. Although thousands of miles away in Canada, I have always known that I can call Ron up at any time and count on him to encourage me no matter how well or badly things might be going at the present moment.

I thank my friend Kris for always encouraging me to finish my degree and for many inspiring discussions about the field of artificial intelligence that we have had. I'd also like to thank Kris and my friend Tamara for the many fun times we have had together during the time when I was writing this thesis, without which I would no doubt have gone insane or never even finished the thesis.

Finally, last but certainly not least, I thank my parents for always believing in me from when I was a small child right up to the present moment. Without their constant encouragement, I would no doubt not have even been in a position to attempt a Ph.D., much less to complete one.

Table of Contents

List of Figures

Chapter 1
Introduction

In time-stressed domains, relying on decision trees developed using classical methods to solve a diagnosis or classification problem can be dangerous.

Consider the problem of a parachutist trying to diagnose a potential malfunction of the main parachute after noticing it seems to be taking longer than usual to open. The parachutist has a pair of diagnostic tests available which he/she can perform. One test is to look at the altimeter for a few seconds to determine the rate of fall, and thereby determine whether he/she has a low-speed or high-speed malfunction. The other test is to look at the main parachute (or lack thereof) and determine whether there is a good parachute, a parachute not properly open (low or high speed malfunction), or no parachute at all (total malfunction). With these two tests, the parachutist can classify the situation into one of four possible situations--a good parachute, a low-speed malfunction, a high-speed malfunction, or a total malfunction. If the parachutist fails to make a sufficiently specific diagnosis, then the parachutist will not be able to respond effectively in time, and a catastrophic outcome will result. Imagine that the prior probabilities of these four possible situations are as follows:

Situation	**Prior**
Good parachute	0.1 (due to the fact that the parachute is slow in opening, this value is low)
Low-speed malfunction	0.4
High-speed malfunction	0.4
Total malfunction	0.1

This being a time stressed domain, there may only be sufficient time to perform *one* of the two available tests before the parachutist is required to act. Hence the first test performed ought to help the parachutist identify such an action.

From an information-theoretic point of view, the best test to perform first would be to look at the altimeter, because this will distinguish between the most likely possibilities--low-speed vs. high-speed malfunctions. The difficulty with this is that it fails to identify an action to perform in either of these potentially urgent situations. If the rate of fall is slow, the parachutist knows there is either a good parachute or a low-speed malfunction; likewise if the rate of fall is fast, there is either a high-speed or total malfunction. But as it turns out, in this domain this does not give sufficient information to act effectively: a low-speed malfunction requires a quite different action from a good parachute, and similarly a high-speed malfunction requires a quite different action from a total malfunction. Looking at the parachute is a far better choice. Upon doing so, the parachutist still will not know whether there is a high-speed or low-speed malfunction, but he/she will have enough information on which to act: if there is either a low- or high-speed malfunction, he/she must release the main parachute and activate the reserve; if there is a total he/she must pull the reserve immediately (saving time); and if there is no malfunction he/she must steer normally.

This simple example illustrates many of the properties of the problem that this thesis is intended to solve, such as:

There is a time-stressed domain in which some form of response is required by a deadline. Within the domain, the goal is to diagnose one of a number of *faults.* Complete diagnosis of a fault would enable a complete solution of the problem, but a partial diagnosis may be much better than no information at all. However, a partial or complete diagnosis made after the deadline is reached is of no help at all. Partial diagnoses consist of information that the fault belongs to one of a particular set but not which specific fault is the correct one.

There are tests available that the agent (in this case the parachutist) can perform to distinguish among the faults that need to be diagnosed. Each test will have two or more outcomes, and each outcome in turn will correspond to some subset of the total set of faults (the subset of faults consistent with that outcome). For example, in the parachutist domain, the "check alternative" test has two outcomes: low rate of fall, and high rate of fall. The low rate of fall outcome is consistent with two faults: "good parachute" and "low-speed malfunction". The high rate of fall outcome is consistent with the other two faults.

There are actions available that can be performed; each action has some value for each fault. For example, in this domain pulling the reserve immediately with a low-speed malfunction can result in a fatal entanglement, so the value of this action is very low for this fault. It is also the case that actions may have value for a class of faults larger than a single fault. An action may have value for a fault without being the optimal action for that fault--for example, an action resulting in injury but not death might fit into this category in this domain.

The deadline may vary depending on the fault--in this domain, low-speed and high-speed malfunctions have very different deadline distributions. However, the agent will always know when it is about to reach a deadline and must take action. The agent's decision on what type of test to perform may depend on knowledge that it has about deadline distributions--for example, any test that takes longer than the expected amount of time until impact is useless in this domain.

More than one outcome of a test may be consistent with any given fault. It is possible that a test will not provide any information about certain faults, or that it will provide partial information (changing the probability) without providing complete information. For example, in this domain, another test which could be performed would be to check body position (whether upright or horizontal). The high-speed malfunction fault is consistent with both outcomes of this test. This thesis will consider tests that provide no information about certain faults, but not tests giving partial (probabilistic) information about faults. It is also possible (in some domains, not this one) that the agent may be able to initiate several tests simultaneously, and will choose to do so if the value of doing so is greater than the cost of performing extraneous tests; this thesis will treat this question.

Although the agent will have very little time to respond once the emergency occurs, the agent will have considerable time to plan out possible responses in advance of the emergency occurring. However, the goal here is not to provide the agent with a universal plan. Rather, it is to provide the agent with a manageable set of faults that it can reasonably plan to respond to. Furthermore, the agent does not execute a complex plan--it is concerned only with finding a single action that it can apply in a situation. It finds the best action possible given its constraints, and then acts.

This thesis will propose and evaluate a solution to this problem based on the concept of *action-based hierarchies*: an action-based hierarchy is a form of decision tree of sets of faults with actions augmented to each node. The agent finds the most specific possible diagnosis--that is, the lowest possible node in the hierarchy--within the time available, and then performs the action augmented to that node.

In chapter 2, the problem will be described more formally. Chapter 2 will distinguish the two different phases--the reactive planning and the reactive plan execution stages of the solution. The primary goal of the thesis is to find a way of constructing reactive plans--sequences of tests and actions to be performed under certain conditions--so that they find as good an action as possible in as short a time as possible--in other words the execution phase is the dominant concern here. However, it is also necessary that the reactive planning phase--the construction of the reactive plans in advance of execution time--not involve so high a time complexity that it is infeasible. In fact, the problem will be formulated in such a way that there is by definition a best possible reactive plan in any given situation. Although it is assumed that the agent has a considerable amount of time to do reactive planning, this time is not infinite--if it were, it could simply search the space of all possible reactive plans until it found the best one. A reactive planning approach involving time complexity better than NP-complete is necessary here.

Chapter 3 will present the approach. The basic idea is to take the existing AI technology of *decision trees* and augment it with the notion of an *action* at each node in the tree to form a new structure called the *action-based hierarchy*. The action at each node is the best possible action for the set of faults that are descendants of that node in the hierarchy. Ideally, it is an action with good value for all those faults; less ideally it may ignore one or two low probability faults in exchange for good value for all the other faults. Because the action is designed to provide good value for a large class of faults, it is not necessarily the best action for every one of those faults. Given the structure of decision trees, the question arises of what heuristic should be used to decide how to expand each node (the attribute selection problem). Chapter 3 will argue that an action-based heuristic is much more natural than the standard information-theoretic heuristics which are more traditionally used to structure decision trees.

Chapter 3 will also describe various enhancements of the approach that are necessary to solve all the aspects of the problem described in chapter 2. For example, the outcomes of a test correspond to sets of faults. When these sets of faults are not disjoint (one fault may be consistent with more than one outcome to a test) it introduces additional complexities into the solution that is used. When the costs of tests are taken into account, it becomes possible to modify the approach to be able to intelligently recommend--or not recommend--the performance of multiple tests at one time, a desirable feature in some domains. Various tradeoffs must also be considered when more than one fault may appear at one time.

In chapter 4, complexity considerations will be discussed. The approach is designed so that time and space complexity of the generated reactive plan is trivial, so the main concern is complexity at the reactive planning stage. With sufficient assumptions, complexity can be analyzed completely. When the problem becomes more sophisticated, it becomes difficult for reasons that will be discussed in that chapter for the problem to be completely analyzed, but examples taken from a real-world domain are given which shed light on the complexity of the reactive planning approach in general. Chapter 4 begins the part of the thesis that evaluates the approach; it is unique in that it is the only chapter to address the reactive planning part of the approach (as opposed to the value of the generated reactive plans).

The next several chapters are devoted to an evaluation of the merits of the approach in generating reactive plans that meet the goals outlined in chapter 2. It should be noted here that casting any real-world problem into the terms described in chapter 2 will necessarily be an approximation to the real-world problem. Thus, if it is claimed that a particular solution to the approximation is "optimal", it is not necessarily the case that it is an optimal solution to the real problem, but only an optimal solution to the approximation. Although the thesis seeks to evaluate how well the approach solves the idealized problems derived from real-world examples, it does not quantify how close these idealized problems approximate the real world. However, it is not intended that this approach will be an agent's only tool in solving any problem: rather the purpose of making the approximations is to enable the agent to make a quick response in cases where it is required to do so. In cases where the agent has more time, it is possible that a completely different solution, involving a more complete (but more computationally costly) approximation to the real world would be appropriate. Even in such cases, optimality is constrained by the limits on available knowledge and the availability of effective actions.

The first of this group of chapters is chapter 5, in which the evaluation is theoretical: what can be said in absolute terms about how well the approach solves the (idealized) problem it is designed to solve? There are two basic results in chapter 5. The first will show that under certain very strict assumptions, the approach provides the best solution to the problem. The assumptions are too strict to be valid in all but a few domains; hence the more empirical results of chapters 6 and 7 that will follow. The second result of chapter 5 shows that a particular part of the approach, known as test promotion, that is actually an add-on to the basic approach, is guaranteed to improve or at least not worsen the value of the solution. Hence if the computational expense at planning time is not prohibitive, running this phase of the algorithm will be desirable.

The results of chapter 5 are quite incomplete, which is what motivates the work of chapters 6 and 7. The idea of chapter 6 is that the inputs to the algorithm that forms the basis for the approach can be assigned randomly, and the algorithm then run. Any reactive plan can be evaluated even without actually deploying it in an actual agent. Since the approach is a variant on the traditional decision tree approach, the value of a reactive plan generated using it can be compared to that of a reactive plan generated using the traditional decision tree approach. By comparing the results, it can be determined whether the approach represents an improvement over existing technology or not. Chapter 6 will perform these experiments for the basic approach and the various enhancements described in chapter 3.

In chapter 7 a real-world domain is introduced: intensive care unit patient monitoring. Domain knowledge has been gathered from this domain and used as the input to the algorithm; the results are presented in chapter 7. Because this is a single problem, however, the results are not statistically significant. Chapter 7 also presents an analysis of the domain which determines the properties of the inputs to the algorithm. Then a new set of experiments is performed which assigns the inputs in a way that approximates that of the medical domain but that also allows for statistical significance.

So far no mention has been made of an actual implementation of an agent that can use reactive plans generated using this approach. That is the goal of chapter 8. Because the analysis in chapters 4-7 is self-contained, there is no need in chapter 8 to analyze the performance of the agent. However, a number of interesting issues come up in the actual implementation of the agent which are discussed in chapter 8. In addition, this provides a good framework to provide an actual example of the approach in action. Although chapter 7 provided an overall evaluation of the approach for the medical domain, it did not provide specific examples of how the approach would apply to specific faults. Chapter 8 provides such an example.

Finally, the research is summarized and potential future work is described in chapter 9, the conclusion.

Two appendices are provided: appendix A gives a programming manual for the implementation described in chapter 8. Appendix B gives detailed information about the medical knowledge base that served as the basis for the experiments in Chapter 7; the medical knowledge was provided by our medical colleagues, and its primary author is Garry Gold.

Section 1.1 -- Related Work

Several bodies of literature connect to this thesis work.

The thesis represents an advance on existing work in anytime planning; this approach trades off inexpensive computation time before the agent is time-stressed for expensive time during a stressful situation. Existing diagnosis literature either does not address diagnosis under deadlines or does so only tangentially by concerning itself with the complexity of diagnosis algorithms. A variety of reactive planning approaches has been taken in the literature, none of which adequately addresses the problem. A similar comment can be made about real-time planning systems. A detailed description of these approaches, with appropriate references and reasons why they do not or only partially solve the problem at hand, can be found in section 2.2.

There has been a significant body of literature devoted to the problem of how one structures a decision tree (the so-called attribute selection problem). In one sense this literature is not entirely relevant to the current thesis because it is designed to solve a different problem than the one described here. However, an overview of this body of work, which relates more to the approach than to the problem, is given in section 3.6.

Finally, with regard to the domain itself, there has been a wide range of work on artificial intelligence in medicine. Within that work, there are two bodies of literature of interest here: those systems that sought to handle real-time situations in a reasonable manner, and those monitoring systems with the intensive care unit as the application domain. A description of these systems, and how the current approach differs from them, can be found in section 7.1.

Chapter 2
The Problem

At the most basic level, we are interested in the problem of identifying and acting on one of a set of *faults*. For the purposes of this thesis, we define a fault as the most specific diagnosis in which an agent is interested within a given domain. Thus, a fault is a property both of the domain and the goals of an agent. For example, a police officer considering handing out a ticket for drunk driving is interested in whether the blood alcohol level exceeds or is under .08%; a physician treating someone for alcohol poisoning has different concerns. From the police officer's point of view, there are two possible faults: BAC under .08%, and BAC over .08%. From the physician's point of view, there may be many different faults and the particular BAC may not be particularly relevant. In other words, there is no objective definition of what a fault is; in this we differ with other writers in the literature who for example view a fault as the failure of one particular component of a mechanical system ([RE87], [DE87]). The use of the term *fault* does imply that a fault in general is something negative which needs to be corrected. This does not preclude including a few "faults" that correspond to normal behavior for completeness, but as a rule we think of faults as undesirable events requiring actions.

For dealing with a set of faults, the agent will have available to it a *reactive plan*. Later in this thesis we will analyze in detail a particular type of reactive plan--the action-based hierarchy--but for now it is most important to realize that a reactive plan consists of a series of actions that an agent may take in order to diagnose a particular fault in real time, together with actions for remedying the fault that it diagnoses. Because the reactive plan is intended to be deployed in real time, it is desirable that it consume a minimum of computational, time and other resources, and that it have an anytime flavor [DB88] to it which will allow it to identify useful actions to perform even in cases where resources do not permit its running to completion.

For the purposes of this thesis, the *reactive planner* that constructs the reactive plan may be quite distinct from the agent that executes a reactive plan. Consider, for example, a pilot faced with making an emergency landing because of engine failure. The pilot has a checklist--a reactive plan--for dealing with this emergency situation. However, it is generally not the case that the pilot designed the checklist. Although the pilot is the agent who will execute the plan, the checklist is likely to be recommended for all pilots flying a particular type of aircraft. This distinction is important because it indicates differences in the desired properties of the reactive plan versus the reactive planner. The reactive planner need not be especially computationally efficient, because it may have substantial time to complete a plan prior to it ever being needed in real time. However, the plan that it generates must be computationally efficient for the reasons noted above. This distinction will guide the design of reactive plans in this thesis.

Partial diagnoses are quite possible in many domains, and it is essential that the reactive plan be able to handle the possibility of making a partial diagnosis. For example, a physician may recommend a transfusion if he/she knows a patient's blood type and that the patient has an anemic condition--even without knowing the specific type of anemic condition the patient is suffering from. Such a recommendation is not necessarily optimal--knowing the particular type of anemia would enable an optimal recommendation--but may be necessary to save the life of the patient. The reactive plan must enable an agent to reach similar goals--finding an optimal action if time permits, but taking an action that is less than optimal if necessary because of a deadline.

We take the view that a partial diagnosis is a set of faults and a complete diagnosis is a single fault. Hence as the reactive plan is executed, the diagnosis will gradually be refined from the set of all faults in the domain to the particular fault that is present in a given situation.

The agent must be able to respond by a *deadline.* We recognize two particular types of deadlines--hard and soft deadlines. A hard deadline is that time by which an agent must have taken action or a catastrophic outcome will result. A soft deadline, which is always either at the same time or earlier than the hard deadline, is that time after which it is better to act than to delay acting while further refining the diagnosis. Notice that although the agent will therefore act upon reaching a soft deadline, it will still continue to perform diagnosis until it reduces the diagnosis down to a single fault: the soft deadline is not the signal to stop diagnosis but rather to start action. We make the assumption that the agent always knows when it has reached a soft deadline; it is not the goal of the current research to identify what the agent's soft deadline is.

An interesting property of hard deadlines is that they do not actually affect the performance of the reactive plan. They do not cause the agent to take additional action, because such action is already being performed by virtue of the soft deadline being reached. However, hard deadlines may play a major role in the *design* of the reactive plan, because in designing a reactive plan, the planner will try to make sure that the best possible action is available to the agent before reaching the hard deadline. Thus, there may be information available to the reactive planner involving the probability distributions for the hard deadlines. In particular, for each fault there will be a deadline distribution which represents the probability of the hard deadline falling in different intervals if the particular fault turns out to be present. The idea here is that although the reactive planner cannot know the exact value of the deadline, it should have some notion that certain faults are likely to require quick responses whereas others allow the agent more time, and therefore construct the reactive plan accordingly.

The hard deadline is measured from the time when the reactive plan is first invoked--that is, a hard deadline of 30 minutes indicates that the agent had 30 minutes after invoking the reactive plan to take action. The agent will generally invoke the reactive plan after noticing that something appears to be going wrong--perhaps it had a regular plan which called for a parameter value to be in a particular range, and the value is outside that range. It is obvious that the invocation of the reactive plan does not necessarily correspond to the time when the fault first appeared, so the reader may ask why we do not measure hard deadlines from the time when the fault appeared. The answer is that the reactive planner cannot make use of such knowledge. If it knows approximately how long the agent will have after invoking the reactive plan to take action, it can construct the plan accordingly. But if the planner only knows how long the agent will have since the fault first appeared, it cannot be sure when the plan will be invoked and therefore will be operating in the dark. Thus, in providing deadline distribution information to the reactive planner, one must be careful to measure deadlines from the time the reactive plan will be invoked--which might be when the first indication of the fault appeared, not when the fault itself appeared.

Just as hard deadlines do not actually affect the behavior of the agent in executing the reactive plan, we do not use soft deadlines in constructing the reactive plan. The reason is that we are most interested in the agent having a good action to perform by the hard deadline. Although it is even better if this action is available by the soft deadline, the planner does not sacrifice possible performance at the hard deadline by striving to perform as much diagnosis as possible before the soft deadline. Hence we have an interesting dichotomy in how deadlines are used: soft deadlines exclusively in executing the reactive plan, and hard deadlines exclusively in designing the plans. The agent executing the reactive plan does not need to know the hard deadline, because the soft deadline is the time by which action is desirable. Similarly, performance is evaluated based on the action available at the hard deadline, so the reactive planner does not need to know anything about soft deadlines.

We have stated that we view a fault as something that needs to be corrected. Because of this, the agent also has a set of *corrective actions* which it can take in executing the reactive plan. Because the agent is executing a *reactive* plan, it is not interested in designing a long sequence of corrective actions: rather it views the execution of a single corrective action as being a solution to a fault. However, the reactive planner recognizes that some solutions are potentially better than others, and so it has a notion of the *value* of performing a corrective action. This is a purely heuristic estimate given either directly by the domain expert or by some well-understood computation within the domain. For the moment, it is assumed to be positive--negative "values" (actually costs of actions) will be discussed below. For each combination of a corrective action and a fault, there is a value representing the value of that action for that fault. The resulting matrix is expected to be rather sparse in that most corrective actions have no value for most faults. For example, filling the tires with air will not fix a broken-down engine, but the reactive planner needs to know this either explicitly or implicitly.

Because of the sparseness of the value matrix, it is not expected that a domain expert would need to fill in the entire matrix. Rather, the domain expert would fill in values only for those actions that have an interesting effect on particular faults, and the rest of the matrix would be automatically filled in with zeroes. Indeed, in a large domain, the entire matrix would not need to be represented explicitly even internally within the reactive planner.

There are also *costs* associated with pairs of actions and faults. Again, the cost is a heuristic estimate of the cost of performing a particular action in the presence of a particular fault. Here we expect that in most cases the cost of performing an action will be independent of the particular fault it is being applied to. This captures the notion that if the action has no value for the particular fault, then the cost of performing it makes it undesirable to perform the action. Because the cost of performing an action is in general independent of the fault involved, again the cost matrix can be provided by the expert by simply providing costs for the actions and then noting exceptions for particular faults.

In practice, we combine the notion of value and cost into a single notion of total value, which represents the difference between the value and the cost. This total value can therefore be negative, and for the remainder of this thesis when the term "value" is used, it is this difference that is being referred to. This will simplify the analysis. The difference between value and cost appears to be the appropriate combining function. If we took the ratio, actions with little value but infinitesimal cost would be more desirable than actions with high value but moderate cost--this is clearly not a desirable situation.

In order to help the agent diagnose one of the faults, it has available to it a set of *tests* which it can perform. Each test has a set of possible outcomes, and each outcome is a set of faults (a subset of the full set). Thus, when a test is performed and an outcome is obtained, the agent knows that the actual fault belongs to the set corresponding to that outcome. If the agent already has information from another test or group of tests that the actual fault falls within some *differential diagnosis*, then the agent can take the intersection of the differential diagnosis with the current outcome to produce a new, smaller differential diagnosis. Gradually by narrowing down the differential the agent hopes to reduce it to a single fault and complete the diagnosis process. The fault sets associated with the outcomes for a given test need not be disjoint--some tests may not provide any information one way or the other about certain faults. In this thesis, the assumption is made that if a given fault is present and this fault is included in more than one outcome of a particular test, all such outcomes are equally likely when the test is performed.

A test also has associated with it a set of *monitoring actions* and *diagnostic actions*. A monitoring or diagnostic action is an attempt by the agent to gain information from the world. Specifically, a monitoring action represents an increase in the rate at which a given parameter, which is always available to the agent at some frequency anyway, is monitored. A diagnostic action is a request to obtain a single value of a particular parameter from the world. Both types of actions have two types of costs associated with them. There are temporal costs--the amount of time that it takes for the action to be performed and the results to come back. Because of the real-time nature of the deadlines the agent is up against, temporal costs are important to the agent. Temporal costs in turn have two components--the time taken in the world for the parameter values to be obtained, and the computational resources consumed by the agent in analyzing the parameter values to determine a particular outcome for a test. The exact value of the temporal costs will therefore vary depending upon the amount of computational resources the agent has to devote to the diagnosis task. There are also physical costs--the amount of physical resources consumed by the action. In general, these two types of costs must be traded off against one another--we could perform all possible actions with great frequency and save on temporal costs, but at great expense regarding physical costs. Similarly, we could save on physical costs by performing only one monitoring or diagnostic action at a time (thereby performing only those actions that are absolutely necessary) but this would increase the temporal costs.

Faults have associated with them a set of *prior probabilities*. Because we assume that when the reactive plan is invoked, at least one fault is present, the prior probabilities must sum to at least 1.0. In the event that we make the *single fault assumption* (see below), the sum of the priors must be exactly 1.0.

It follows from the above description of the basics of the problem that the reactive plan will be executing a form of *anytime algorithm*. At various points in time, it will request that tests be performed, and the results of these tests will result in a refinement of the differential diagnosis. However, for any differential diagnosis, it is possible to find a corrective action that is better than all others, on average, for this differential. Ideally, such an action would be one that had substantial value for most of the faults in the differential, and as a result it is desirable that the reactive plan be constructed in such a way that it is likely to find differentials along the way that have good corrective actions for most of their faults. The algorithm is anytime in the sense that as the differential is refined, the value of the best corrective action available is likely to improve as it can be made more specialized for particular faults. When the soft deadline is reached, the corrective action associated with the current differential can be recommended and performed.

The problem can be constructed either making, or not making, the *single fault assumption*. The above description tends to make the single fault assumption. A number of things change somewhat when the single fault assumption is not made. Most importantly, whereas tests previously were assumed to return only a single outcome when performed, now tests may have multiple outcomes per performance. Whereas the previous semantics of a test outcome was that the one and only fault was in the outcome fault set, now the semantics of a set of test outcomes is that at least one fault from each outcome set is present. Another difference is that it becomes necessary to represent values of actions for a set of faults. For example, if either engine on a twin-engine plane fails individually, flying to a nearby airport may be the most desirable action. But if both engines fail, then making an emergency landing in an empty field will be the most desirable. The matrix representation described earlier will not be sufficient to capture this domain knowledge.

Section 2.1 -- Formal Statement of the Problem

The previous section gave an intuitive statement of the problem we are interested in. In this section we will state the problem more precisely. The reactive planner is given a 9-tuple with which to construct a plan:

$$<F, P, D, A, V, T, O, C, S>$$

These are defined as follows:

F -- the set of faults.

P -- the set of prior probabilities. *P* is a function mapping *F* to the interval [0,1].

D -- the set of deadline distributions. *D* is a function mapping *F* to the set of functions, D_f, such that

$$\int_0^\infty D_f(t)dt = 1$$

Here the semantics are that if $D(f) = D_f$, then the probability that the hard deadline for fault *f* lies in the interval $[t_1,t_2]$ is:

$$\int_{t_1}^{t_2} D_f(t)\,dt$$

A -- the set of actions.

V -- the matrix of values. *V* is a function mapping *FxA* to the interval [-1,1]. The semantics of this will be discussed in more detail below.

T -- the set of tests.

O -- the set of outcomes to tests. O is a function mapping T to sets of sets. For example, if $O(t) = \{ o_1, o_2, ..., o_k \}$ is a set of outcomes, then each o_i is a subset of F corresponding to one possible outcome of the test t.

C -- the set of temporal costs to tests. This is a function mapping T to the set of nonnegative real numbers.

S -- the set of physical costs to tests. This is a function mapping T to the set of nonnegative real numbers.

It will be noted that a couple of simplifications have been made from the intuitive description of the problem given in the last section. First, the notion of action-value and action-cost have been collapsed into a single notion of action-value (V). The rationale is that we can assign an action-value in the interval [0,1], and then an action cost in the interval [0,1], and finally take the difference to give a combined action-value in the interval [-1,1]. Also, the notions of monitoring and diagnostic actions have been removed. As noted in the last section, a test consists of a set of monitoring and diagnostic actions. However, the interesting properties of a these monitoring and diagnostic actions can be capture in the temporal costs of tests and physical costs of tests using the combining functions described in the last section. It will not be a topic of research in this thesis to describe how to construct tests out of the more atomic monitoring and diagnostic actions, so for the purposes of the analysis we can leave it at that.

We now describe the goal of the agent vis-a-vis the reactive plan. The agent's goal, roughly speaking, is to recommend a series of tests

$$<t_0, T_0>, <t_1, T_1>, <t_2, T_2>, ...$$

where $t_0 \leq t_1 \leq t_2 \leq ...$ are the times at which tests $T_0, T_1, T_2, ...$ are recommended. All times are measured from the invocation of the reactive plan. When making its recommendation at time t_k, the agent will have available to it the outcomes of all tests that have come back already-- that is, all outcomes to tests T_i, $i < k$, where $t_i + C(T_i) \leq t_k$. The agent must also recommend (and perform) at the soft deadline some action a_s, and at the hard deadline some action a_h. The agent is assumed to know when the soft deadline is and to recommend and perform an action at that point, but it continues to perform diagnosis after reaching the soft deadline unless and until it reaches a complete diagnosis. As mentioned in the previous section, the agent's performance is evaluated on the basis of the value of the action a_h, not on the basis of the action a_s. Also, we assume that the agent will know when it has reached the soft deadline and be able to take action appropriately.

The notions of hard and soft deadlines represent an approximation to the real situation, which is that the value of performing actions decays over time. A complete analysis would take this decay as well as the cost of performing actions into account and thereby determine the optimal time to perform an action. The reason this complete analysis is rejected is that it would require that one perform this computation in real time. Such run time computation is intended to be avoided by this approach; hence the situation is approximated with hard and soft deadlines. Hard deadlines at planning time assume value of action decaying to zero at a single point in time (the hard deadline). Soft deadlines at run time allow some incorporation of a more fluid notion of value of action, albeit through the agent knowing the soft deadline by some means outside of this approach. Should the agent have no such means available to it, it could always simply use the hard deadline in place of the soft deadline.

Because the analysis of the constructed plans is based on hard deadlines, for the remainder of this thesis whenever the term "deadline" is encountered without qualification, it will be referring to the hard deadline. Thus, for the purposes of evaluation of the approach soft deadline is assumed to be equal to hard deadline. Therefore, no specific claims are being made about the usefulness of the soft deadline concept beyond the fact that clearly it is sometimes desirable to perform action in advance of its being absolutely necessary. Hence if the agent knows (by some means outside the scope of this approach) in a particular case that this is desirable, it should take advantage of this knowledge.

The action a_h is a function of the time t at which the hard deadline occurs and therefore may be written as $a_h(t)$. The function should depend on two things--the particular fault that occurs and the outcomes to the various tests (noting that in the case of multiple outcomes including the same fault, all are equally likely). Letting f denote the fault and o denote the various possible outcomes to the tests, we can see the value function is actually a function of three variables: $a_h(t, f, o)$. This may be rendered in numerical terms as a step function by defining:

$$V(t, f, o) = V(a_h(t, f, o), f)$$

This function intuitively represents the improving value of the action obtained to date during a single invocation of the reactive plan. By taking the mean over all outcomes of the test consistent with this fault, we can reduce this to a function of two variables: $V(t, f)$ which should also be a step function, albeit with many more steps. Now we can reduce this to a function only of the fault itself by weighting this value function by the deadline distribution function for this fault:

$$V(f) = \int_0^\infty D_f(t) V(t,f) dt$$

Finally we may compute the overall performance of the agent by calculating the weighted sum over all faults:

$$V = \sum_{f \in F} V(f)\, P(f)$$

This number, V, represents the overall performance of the agent on this reactive plan. Two different reactive plans may therefore be compared by computing different values for V and comparing. Intuitively, what we have done is take the various actions recommended by the agent at different points in time and determine the values of those actions for a particular fault. These values are then plotted on a graph and then the average is taken of the different performances that are possible for a single fault because of uncertainty in test outcome. We then determine the average performance over time of the agent on this fault by computing the integral weighted by deadline distribution. Finally, we compute the average performance of the agent over all faults by taking the average of these functions weighted by probabilities of the faults. This is illustrated graphically in Figure 1. This shows the improving performance of the value of the action recommended for a fault over time as the action changes from $A0$, $A1$, $A2$, through $A3$. Because in this case the hard deadline is assumed to have a uniform distribution between 2.5 and 3.5, only the values of the actions recommended at those time points are relevant, and the integral computing overall performance is calculated over that interval.

So far the analysis has taken into account only the value of performing actions. What of the cost of performing tests as described in the last section? The temporal cost of performing tests is taken into account when we compute the time axis of these graphs. The physical cost of performing tests will itself be a step function of time, in this case guaranteed to be non-decreasing, which can be computed for a single invocation of the reactive plan as follows:

$$S(t,f,o) = \sum_{t_i \le t} C(t_i)$$

As with values, we can compute the average over all test outcomes consistent with the particular fault to give $S(t,f)$. Now we can compute the weighted integral over time to give:

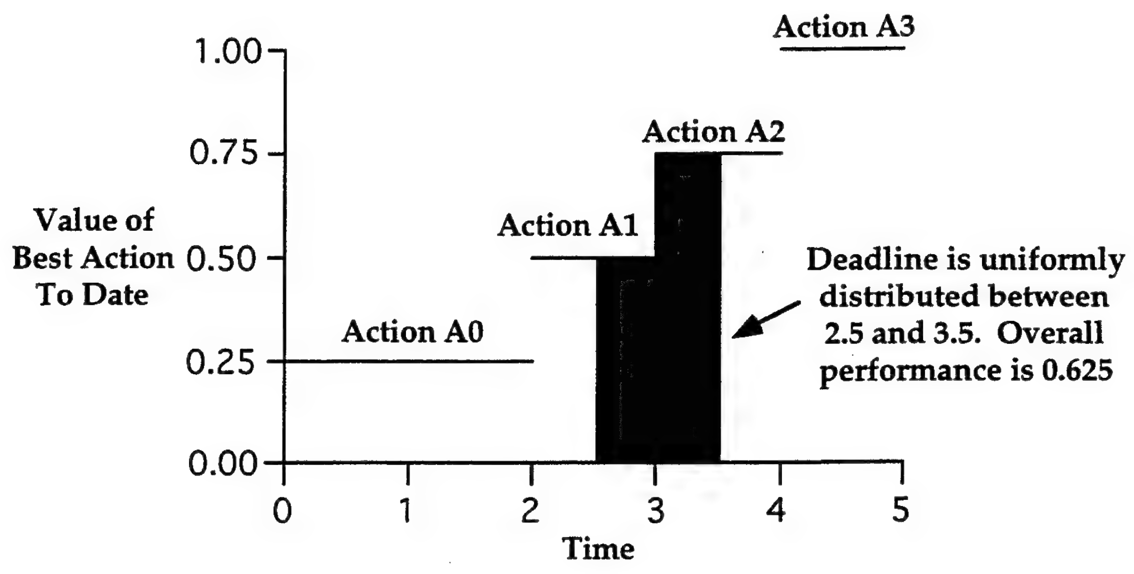


Figure 1:
Reactive Plan Performance on a Single Fault

$$S(f) = \int_0^\infty D_f(t)S(t,f)dt$$

Finally, we can compute the overall average physical cost of this reactive plan by computing the weighted average over different faults:

$$S = \sum_{f \in F} S(f)P(f)$$

The true performance of a reactive plan may then be computed by determining the value-cost difference V-S.

Our goal in this thesis is not to directly produce reactive plans. It is, rather, to produce a reactive planner that will generate good reactive plans. Thus, we will have a space of problems $\{<F, P, D, A, V, T, O, C, S>\}$. By computing the average performance of the reactive plans produced by different reactive planners for the same problem, we will be able to compare reactive planners. It is our goal in this thesis to propose and evaluate a set of reactive planners on just this criterion.

This completes the statement of the problem for the single fault case. The multiple fault case is similar but a couple of changes are necessary to the way the problem is defined:

P-- Prior probabilities remain defined as before. However it is no longer necessarily the case that the sum of the priors is 1.0.

V -- The matrix of values now maps $F \times 2^F \times A$ to the interval [-1,1]. Semantically, this gives the value of a particular action (from *A*) for a particular fault (from *F*) in the presence of a particular set of other faults (from 2^F). We expect this matrix to be very sparse, including entries only for interesting combinations of faults. The actual value will be for a given fault-action pair the minimum of all values included in the matrix that are maximal subsets of *F*.

O-- It is now possible for tests to have multiple outcomes occur for a given invocation of the test. As before, we assume that all minimal sets of outcomes that are consistent with a given set of faults will be equally likely (and all other sets of outcomes do not occur).

The agent in executing the reactive plan may be working on more than one partial diagnosis at a time, so it will now need to compute the best action on the fly. We will show in Section 3 that if the size of the matrix of values is reasonable, this computational complexity should not be unreasonable for a small number of faults. The reactive plan is just not intended to deal with large numbers of faults occurring simultaneously, so eventually performance must degrade if the number of faults it is asked to deal with becomes too large.

We note that with the above definition of the problem, there are two possible ways that the agent could diagnose multiple faults with its reactive plan. It can have tests come back with multiple outcomes, which will lead to multiple diagnoses of faults. Alternatively, it could be that all relevant tests that might distinguish between a pair or group of faults have been performed, with single outcomes, but there remains more than one fault in the differential diagnosis. The agent does not have the opportunity to choose which path it will follow to diagnosis of multiple faults, but we shall see that there are advantages and disadvantages to each path.

Also, we note that although we now assume that the sum of the prior probabilities may be greater than 1.0, we have not taken into account the probabilities of combinations of faults occurring at once. We will examine later in this thesis whether this assumption seriously harms us in terms of performance of the agent.

Section 2.2 -- Related Work

The most obvious attempted solutions to this type of work to appear in the literature are the *anytime planning* work of Dean and Boddy [DB88], the decision theoretic analysis of Horvitz [HO87], and the concept of *bounded rationality* elucidated by Russell and Wefald [RU91]. Anytime planning involves the idea of an *anytime algorithm* which returns an answer regardless of how long the algorithm runs for, but the value of the answer improves over time. Their contribution is to analyze the properties of anytime algorithms in general but not to propose a particular anytime algorithm; this thesis will follow in this general framework but will present a particular anytime algorithm whose properties can be analyzed. Horvitz and Russell and Wefald do a good job of analyzing the problem from a theoretical point of view. However, the goals of these researchers all differ somewhat from the present research in that they assume that deliberation or metareasoning time is nontrivial. In other words, they devote significant computational resources to deciding what to do, rather than actually doing it. In the current research it is an important goal, because of the urgent nature of the domains the agent is expected to operate in, to keep metareasoning time to a trivial level. Any computational resources used by the agent should be devoted exclusively to analyzing the results of tests, not deciding what test to perform next. We are willing to incur a potentially high computational expense at the reactive planning stage in order to save a much smaller amount of time in executing the reactive plan itself. The approaches mentioned above are unable to make this tradeoff because they assume that a single agent does both deliberation and acting, and so time spent deliberating is time lost acting.

Although we characterize this work as a diagnosis problem, the connections between it and the existing diagnosis literature do not seem too close. Classically diagnosis has been cast as a problem in first-order logic [RE87]. This approach requires that one have a model of the domain in order to do diagnosis. As such Reiter offers one significant advantage over our work--the ability to do first principles diagnosis--and has one significant disadvantage--meeting a real-time deadline is impossible. Reiter also shows that diagnosis is greatly simplified when the *single fault assumption* is made--a conclusion that we agree with although we do provide a mechanism for extending our approach to the multiple fault case.

On the subject of multiple faults, the seminal paper is [DE87]. De Kleer and Williams take the approach that diagnosing multiple faults is computationally expensive and so they provide a number of techniques for reducing the computational complexity of the search. For example, they suggest beginning the search with small potential conflict sets and then expanding the conflict sets to produce minimal sets that actually explain the flawed behavior of the system. This works when doing diagnosis from first principles but seems to be a hard idea to apply to our problem. Our agent starts with a large differential diagnosis and then gradually works it down to smaller sets of faults. However, we agree with their basic observation that if the number of possible faults is small, then coping with their interactions remains manageable, but that we cannot expect the problem of diagnosing all possible subsets of the set of all faults to be computationally tractable.

Other researchers such as Cooper [CO90] have addressed the problem of computational complexity of diagnosis without looking at meeting real-time deadlines. For example, Cooper has shown that inference in belief networks is NP-hard. The response to this is to attempt to control the search; the TOP N algorithm [HE91] can be used to generate only the *n* most likely diagnoses. In addition to the lack of meeting real-time deadlines, this approach differs from ours in the use of a Bayesian belief net to represent uncertainties. The current work is not an attempt to deal with uncertainty, although it might be a worthwhile possible extension to attempt to do so.

Other approaches to diagnosis include model-based reasoning [DA88], heuristic classification [CL85], qualitative reasoning [KU86], [FO84], decision trees [QU83], and medical diagnosis [SH76]. None of these approaches attempt to deal with the problem of diagnosis in real time, and so are of limited usefulness to us as is, although an adaptation of Quinlan's decision trees will prove to be central to our approach as we describe it in the next chapter.

Another large body of literature of interest to us is the work on *reactive* and *real-time planning*. Before describing the relation of this work to the present paper, we should define what we mean by these terms as different authors may use them for different purposes. For us, reactive planning is the task of planning *in advance* response to situations that may arise in real-time for which there will not be adequate time to respond if we have to do substantial planning at run time. The work in this thesis fits our definition of reactive planning.

The ultimate example of reactive planning is *universal planning* [SC87] where the reactive planner attempts to enumerate in advance exactly what to do in *all* situations the agent may encounter. Ginsburg [GI89] has shown that in general universal planning is not computationally tractable, although it may be in specific cases. We do not attempt to enumerate all possible situations in advance. Rather, we enumerate a set of faults [DA94] to which we expect it to be desirable to respond to (including, for example, those that are likely to be *critical*, but not so critical that response is hopeless), and then plan to respond only to that set of faults. We therefore are attacking a narrower problem than Schoppers did, but with greater chance of success.

A much less extreme example is *triangle tables* [FI72] where the agent has a plan to execute, but also has a series of preconditions to each step in the plan, so that it knows when the situation has changed unexpectedly so that the plan no longer applies and thereby the agent can replan. This work represented one of the first efforts in reactive planning, but it is limited to recognizing when a plan no longer applies as opposed to providing good actions to perform in such a case.

Brooks' *subsumption architecture* [BR86] is part of this general body of work, although he would probably not classify his work as reactive planning. The approach involves dividing a robot's task into a number of layers, with the lower layers providing simple abilities such as avoiding objects, and the higher layers providing more complex abilities such as identifying objects. This allows reactivity at the lower levels while still permitting more complex operations at a high level. Perhaps Brooks' most controversial claim is that there is no need for a central control structure in a robot. Although Brooks has provided convincing evidence that his approach works for a simple robot with two or three layers, it is not clear that it can scale up.

Kaelbling and Rosenschein have authored a number of papers on *situated automata theory* ([KA87], [KA88], [KA90], and [RO89]). This theory is based on the assumption that an agent can accurately track and represent the state of the world in real time. The basic idea that they have is that they can provide guaranteed response in a single (constant time) cycle. This seems to imply that their approach would be useful if all deadlines were constant (or at least multiples of a constant) but in the case of our problem, not only do the deadlines vary in a much richer way, the costs of tests may not correspond exactly to cycle time either, so fitting our problem into their approach seems difficult. Their approach seems far more likely to be successful when agent behavior can be mapped into *levels of competence* such that at each level response is required within a constant cycle time. It simply is not obvious that our problem can be divided up in this way.

Chrisman and Simmons [CH91] introduce the notion of *sensible planning* which brings decision theory into the analysis of reactive planning. A robot often needs to perform sensing actions in order to diagnose the state of the world. Chrisman and Simmons use decision theory to determine which of several possible *sensing procedures* involves the minimal expected cost. For simple worlds, their approach is probably quite effective, but they have no notion of meeting real-time objectives--they are simply interested in using the best possible sensing procedure.

A number of approaches have been taken to *real-time planning* in the literature. Of these, perhaps Dean et.al. [DE93] attack the problem closest to the work in the present thesis. They compute a *policy* (a mapping from states to actions) on the basis of which they can maximize the future discounted value, averaged over time, of the world state. This problem is similar to ours in the sense that states in their view correspond to differential diagnoses in ours, and a policy corresponds to a decision of ours to perform a test in a certain situation (their notion of *action* corresponds to our notion of *test*; our notion of *action* can be used to compute their notion of the *value of a state*). However, our problem differs from theirs in several important respects. Firstly, all possible states in our world correspond to the set of all subsets of our fault set; for a reasonable number of faults this would be intractable. Second, there seems to be an unstated assumption that the time to progress from one state to another is constant in their domains; in our domains it is not. Finally, they seem to require the specific mathematical properties of the value discounting function in order to make their algorithm tractable. This function corresponds roughly to our notion of deadline distribution, which means that their approach only works for one particular type of deadline distribution.

Hendler and Agrawala [HE90] describe a system in which they unify a *dynamic planning* system with a real-time operating system (MARUTI). Dynamic planning systems are able to both react and plan, and the amount of time devoted to the two tasks itself varies dynamically based upon the nature of the task. Dynamic planning is designed to address the main flaws in both classical planning and reactive planning: it does not assume that a complete plan can be laid out in advance with no unforeseen difficulties, and yet it also does not require anticipation of all possible situations in advance, either. Hendler criticizes existing planning systems on the grounds that they assume that the planner has complete knowledge of the world. He therefore provides a reaction component to handle those situations that the planner could not foresee in advance. However, we go a step further by not assuming that even the reaction component has complete knowledge of the world. But we do not provide any dynamic replanning either, although we anticipate that an agent using our reactive plans will also have other reasoning techniques in its arsenal that will be capable of dynamic replanning.

Chapter 3
Action-Based Hierarchies

In this chapter we will describe the approach, which we call *action-based hierarchies*, that we take to solving the problem outlined in the last chapter. We will first describe the approach as it applies to a simplified version of the problem, for illustrative purposes, and then extend the approach to deal with the complete problem. We will also discuss the feasibility of the approach along several dimensions. First, the time complexity of the reactive plan must be very low. Second, the space complexity of the reactive plan must at least be manageable. Finally, the time complexity of the reactive planner, although not central to our goals, must be low enough to permit the use of our approach (and possibly to allow some dynamic replanning).

The simplified version of the problem that we will solve first makes the following set of assumptions:

- Tests have no physical costs.
- No more than one test can be in progress at a time. If the agent performs a test, it must wait for the outcome of that test before performing the next test in its reactive plan. The first simplifying assumption tends to vastly increase the number of tests that can be performed; this assumption limits the number.
- Deadline distributions are not taken into account. As such we cannot make absolute claims about the performance of a reactive plan unless it performs better than its competitors for *all* possible values of the deadline. Otherwise we can merely observe which deadline values one reactive plan is better for, and which other deadline values another plan is better for.
- The reactive plan is invoked as soon as the fault appears (the previous assumption requires this assumption to also be made).
- The fault sets associated with the outcomes of each test are disjoint (that is, for any given fault, there is exactly one possible outcome of each test).
- The single fault assumption is also made.

The basic data structure that is used to represent a reactive plan is known as an action-based hierarchy. An example of an action-based hierarchy is shown in Figure 1. Action-based hierarchies have the following properties:

- The basic structure of an action-based hierarchy is similar to that of a decision tree.
- The leaf nodes of the hierarchy correspond to individual faults (unlike in a decision tree where they may correspond to classes of faults one wishes to diagnose).
- Each higher-level node has associated with it a set of faults that is the *union* of the sets of faults associated with its children.
- The top node has associated with it the set of all faults
- Each node has associated with it a corrective action, which ideally will have substantial benefit for all faults in the associated fault set, but if no such action exists, may simply have substantial benefit for the most likely such faults.

The central idea behind an action-based hierarchy is that by identifying actions with sets of faults, if the agent only has time to make a partial diagnosis, it will be able to perform the action associated with the set of faults related to that partial diagnosis, and derive the benefit associated with that action. The goal, ideally, is still to perform complete diagnosis, but it is also necessary to be able to provide good response in a suboptimal situation where complete diagnosis is not possible.

As shown in Figure 1, there is also associated with each non-terminal node in the hierarchy a test from the set T (defined in the last chapter). Suppose that node n has associated with it a test t, and $O(t)$ (the set of outcomes of the test t) = $\{ o_1 , o_2 , \ldots , o_k \}$, and that node n has an associated fault set F_n. Then n will have up to k children in the hierarchy, one corresponding to each outcome o_i for which $o_i \cap F_n$ is non-empty. We can thereby formally define an action-based hierarchy as follows:

- The hierarchy itself consists of a pair $<R,N>$ where R is the root node of the hierarchy and N is the set of all other nodes.

- Each node n in the hierarchy is a 4-tuple $<L, t, a, F_n>$ defined as follows:
  - L is the set of children nodes of this node. In the case of a terminal node, this would be empty.
  - t is the test associated with this node (or the empty set in case L happens to be empty and we are dealing with a terminal node).
  - a is the action associated with this node (how this is computed will be explained below).
  - F_n is the set of faults associated with this node.

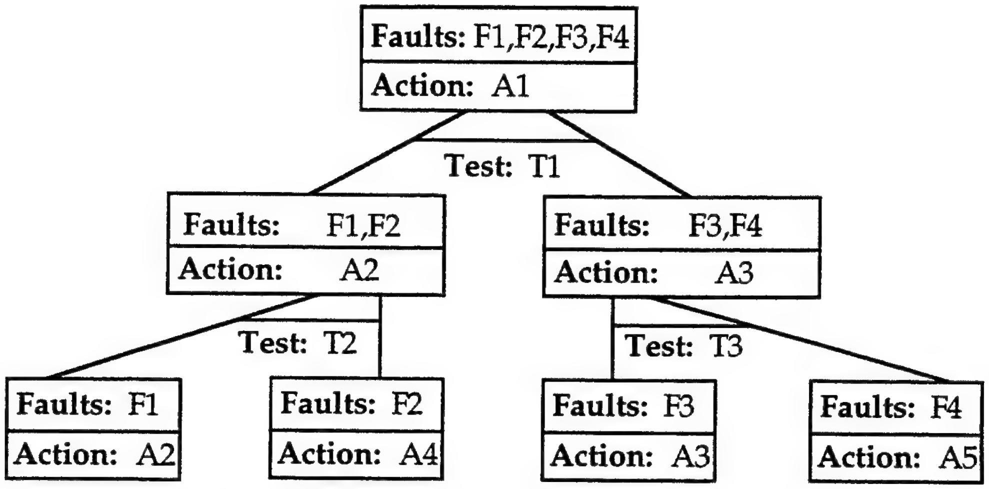


Figure 1: Sample action-based hierarchy

The property given above relating a node to its children may be formally stated as follows: If a node $n = <L, t, a, F_n>$ has children $L = \{n_1, n_2, ..., n_i\}$, with each child n_i having the form $<L_i, t_i, a_i, F_{n_i}>$, and the test t has k outcomes $\{ o_1, o_2, ..., o_k \}$, then for each o_k for which $o_k \cap F_n \neq \emptyset$, there exists exactly one child node n_j for which $F_{n_j} = o_k \cap F_n$.

Having defined the relationship between a node and its children, we now state the conditions that define the action for each node. Specifically, we always associate with a node in the hierarchy the action that gives the best expected value for the set of faults associated with that node in the hierarchy. In formal terms, we can define the value of an action for a set of faults by extension from the definition for a single fault:

$$V(F_n,a) = \frac{\sum_{f \in F_n} P(f)\,V(f,a)}{\sum_{f \in F_n} P(f)}$$

where $P(f)$ denotes the probability of fault f. Next, we can formally define the property that the best action, a_n, at a particular node in the hierarchy must have--if the node is $n = <L, t, a_n, F_n>$, then

$$V(F_n, a_n) = \max_{a \in A} V(F_n,a)$$

In this case we write

$$a_n = a\,(F_n)$$

Having defined the best action for a node, and thus the action-based hierarchy, we can now specify the algorithm that constitutes the reactive plan that the agent will use:

Algorithm 1 -- Reactive Plan

Step	Description of Step
1	Set the *current best hypothesis* to the root node, *R*, in the hierarchy.
2	Perform the test associated with the current best hypothesis.
3	If the test result does not come back before the deadline is reached, go to step 5. Otherwise, modify the current best hypothesis to the child of the current best hypothesis corresponding to the outcome of the test that occurs.
4	If the current best hypothesis is a terminal node, recommend the action associated with that node and stop. Otherwise, go to step 2.
5	Recommend the action associated with the current best hypothesis and stop.

Thus, the reactive plan is completely determined once the agent knows the hierarchy structure. This property of the reactive plan is designed to meet the criterion described in chapter 2 that the computational resources consumed in executing it should be minimal. Given this property of the hierarchy structure, the reactive planner needs to be able to structure the hierarchy. It is structured using the following algorithm:

Algorithm 2 -- Hierarchy Structuring Algorithm
(The Reactive Planner)

Step	Description of Step
1	Start at the top of the yet-to-be-built hierarchy, associating the set of all faults with this top node.
2	Pick a leaf node in the hierarchy in DFS (depth-first search) order, or stop if there are no more leaf nodes to expand. Associate with this leaf node the action that has the highest expected value for the set of faults associated with this node.
3	If the leaf node cannot be expanded further, go back to step 2 and pick another leaf node.
4	Find all tests relevant to the set of faults associated with the current node.
5	If no tests were found in step 4, go back to step 2 and pick another leaf node.
6	Determine which test found in step 4 is best according to some heuristic.
7	Expand the current node with one child corresponding to each possible outcome of the test found in step 6, and the associated fault sets suitably adjusted.
8	Go back to step 2 and pick one of the children of the current node.

The structuring described above is very much like that used in structuring a decision tree. The only part of this algorithm that is not completely specified is in step 6--the heuristic used to determine which test is optimal. There are several possible *hierarchy structuring heuristics* which could be used.

Section 3.1 -- Hierarchy Structuring Heuristics

A hierarchy structuring heuristic is a function that maps tests to the real numbers, within the context of a particular node in the hierarchy, such that the larger the result, the more desirable it is to perform at the node. The simplest heuristic that could be used is the information theoretic heuristic used in structuring decision trees [QU83]. Specifically, let the set of faults associated with the current node be F_n, and compute the following functions of the test t:

$$P(o) = \sum_{f \in o \cap F_n} P_W(f)\ ,\ I(o) = \sum_{f \in o \cap F_n} \frac{P_W(f)}{P(o)} \log \left(\frac{P_W(f)}{P(o)}\right)$$

and

$$P_W(f) = \frac{P(f)}{\#\{o \in t:\ f \in o\}}$$

We then compute the heuristic as follows:

$$\sum_{o \in t} P(o) I(o)$$

This heuristic maximizes the information content gained as one moves from a parent to child node in the hierarchy. Hence it would be expected, in general, to take a minimal number of tests to move from the root node to a terminal node in the hierarchy. However, it does not take into account either the cost of performing tests, or the value of the best action available at the intermediate nodes in the hierarchy. Thus, we would expect that we could do better--in terms of the evaluation criteria proposed in chapter 2--by using a structuring heuristic that does take these factors into account.

Therefore, we propose an *action-based* hierarchy structuring heuristic as follows:

$$\frac{V(t,F_n)-V(F_n)}{C(t)} \text{ where } V(F_0) = \max_{a \in A} V(F_0,a)$$

for any set of faults F_0 gives the definition of value for a set of faults, and

$$V(t,F_n) = \frac{\sum_{o \in t} V(o \cap F_n)P(o)}{\sum_{o \in t} P(o)}$$

It will be noted that this structuring heuristic takes into account only temporal costs of tests, not physical costs. As such, it relies rather heavily on a particular assumption made at the beginning of this chapter--that no more than one test can be in progress at a time. If more than one test were allowed to be in progress at a time, and physical costs of tests were not taken into account, then the agent would be advised to perform as many tests as quickly as possible, as they have no intrinsic cost. In later sections we will relax this assumption at the same time as we take physical costs into account.

This gives two possible heuristics for structuring the hierarchy; there are others which will be introduced later in this chapter. One can hypothesize about the behavior of an agent executing a reactive plan derived from either of these heuristics. One would expect the agent that is using the action-based hierarchy to find actions of higher value for shorter values of the deadline. However, the information-theoretic hierarchy would converge faster on terminal nodes, so for higher values of the deadlines this heuristic would be better. Another hypothesis could be that the overall performance (when integrated over all values of the deadline) is likely to be better for the action-based hierarchy. As we will see, it will be difficult to prove these hypotheses formally except in a very specialized case, but experimentally we can and will test these hypotheses later in the thesis.

At this point a word on nomenclature is appropriate. An action-based hierarchy is really a type of decision tree. However, because it differs in two important respects from classical decision trees--there are actions associated with the intermediate level nodes and the structuring heuristic is different--and the problem that it is intended to solve is different, we prefer to not use the term decision tree to describe this structure. The term *information-theoretic hierarchy* used in the preceding paragraph is closer to the classical decision tree in that the structuring heuristic is the same, but there are still actions associated with intermediate level nodes and it is intended to solve a different problem than the classification problem solved by classical decision trees.

Section 3.2 -- Deadline Distributions

The previous sections described the solution to the simplified problem. In this section and following sections, we discuss approaches for the more complete problem. In particular, we relax the third assumption stated at the beginning of this chapter. The action-based heuristic in the last section was a greedy approach designed to maximize the value gained versus cost ratio at each step in the hierarchy structuring algorithm. In taking into account deadline distributions, the planner should use a heuristic that has several properties:

- If the deadline for a particular fault is so short that no test can make meaningful progress in diagnosing it before the deadline, it is not worth taking into account the value of the actions that are obtained for that fault.

- If the deadline for a particular fault is short--sufficiently short that there is time for only one or two tests before responding--then the performance of the hierarchy at the upper levels will be more significant for that fault.
- If the deadline is very long, then the action-based heuristic we use may not be the most appropriate. The information-theoretic heuristic, which usually moves one to a goal node more quickly than other heuristics, would be the most desirable one to use in this case.

We will first give a heuristic that takes the first two criteria into account and then expand to the third. We first need to build up a set of evaluation functions that take deadline distributions into account. Whereas previously we used the letter V for all such functions, we will now use V_d to denote an evaluation function that depends on deadline distributions. Also, for any given node n in the hierarchy, we will denote the elapsed time in getting to that node by E_n. We thereby define the value of an action for a fault as follows:

$$V_d(f,a,E_n) = \int_{E_n}^{\infty} D_f(t)\, V(f,a)\, dt$$

Intuitively, an action only has value if the agent has not yet reached the deadline, so in determining the expected value for an action, we need to compute the probability that the deadline will not yet have been reached. As before, we can define the value for a group of faults F_n:

$$V_{Gd}(F_n,a,E_n) = \frac{\sum_{f \in F_n} P(f)\, V_d(f,a,E_n)}{\sum_{f \in F_n} P(f)}$$

We can then denote by $a_d(F_n,E_n)$ the action a that maximizes the above function. Next we can define the value of the best action for a fault over a particular time interval as follows:

$$V_I(f,a,E_n,i) = \int_{E_n}^{E_n+i} D_f(t)\, V(f,a)\, dt$$

We can compute this over a set of faults:

$$V_{GI}(F_n,a,E_n,i) = \frac{\sum_{f \in F_n} P(f)\, V_I(f,a,E_n,i)}{\sum_{f \in F_n} P(f)}$$

and then make it independent of the particular action by using the best action:

$$V_I(F_n,E_n,i) = V_I(F_n,a_d(F_n,E_n),E_n,i)$$

The resulting heuristic that takes into account deadline distributions can now be given:

$$\frac{V_{Gd}(t,F_n,E_n)+V_{GI}(F_n,E_n,C(t))-V_{Gd}(F_n,E_n)}{C(t)}$$

where

$$V_{Gd}(F_n,E_n) = \max_{a \in A} V_{Gd}(F_n,a,E_n)$$

for any set of faults F_n gives the definition of value of a set of faults after a particular time interval E_n, and

$$V_{Gd}(t,F_n,E_n) = \frac{\sum_{o \in t} V_{Gd}(o \cap F_n,E_n)P(o)}{\sum_{o \in t} P(o)}$$

Although developing this heuristic required a fair bit of mathematical machinery, the actual intuition behind it is quite simple.

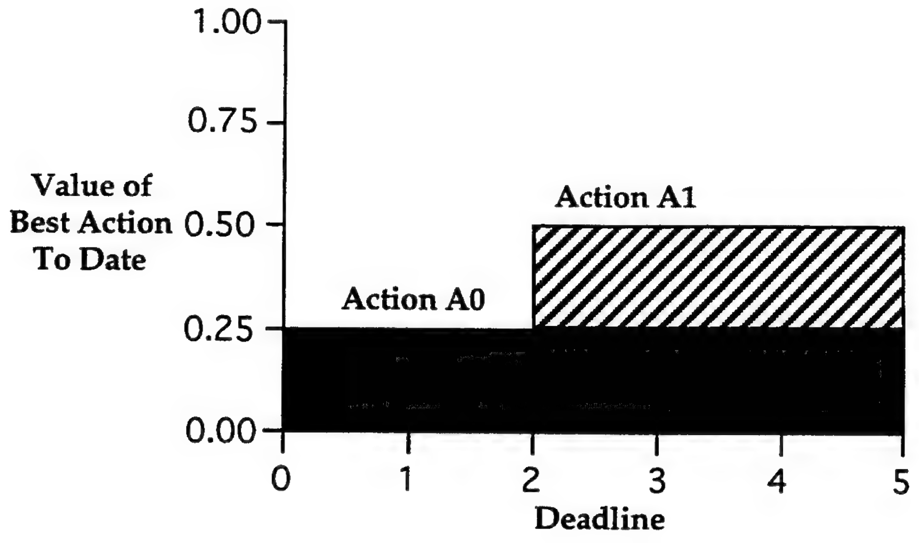


Figure 2:
Deadline Distribution Affects Hierarchy Structuring

Figure 2 shows how this analysis works. This figure shows hierarchy performance on a single fault. The deadline is assumed to be uniformly distributed between 0 and 5. Initially, there is just one root node in the hierarchy, and the action associated with that node, A0, has value 0.25 for this particular fault. The solid black area represents the value of this hierarchy for this fault. However, if we perform a test that takes time 2 to come back, we will get a better action, A1, for this particular fault. Now the value of the hierarchy is given by both the black and striped areas shown above. The structuring heuristic that we use is the *difference* between these two areas, shown by the striped area, divided by the cost of the test. We note how this takes deadline distributions into account: had the deadline been uniformly distributed between 0 and 1.5 (not 5), then there would be no striped area at all in the above graph, and hence the test would have zero value for this particular fault. The above mathematical formulae compute the striped area, averaged over all possible faults.

As mentioned earlier, this heuristic takes into account two of the three criteria mentioned at the beginning of this section. Its major drawback is that it is an essentially greedy approach which does not take into account the possibility that the deadline may be far in the future and immediate returns are less important than long-term results. However, we already have a heuristic that takes this into account--the information-theoretic heuristic mentioned in the last chapter. So we have the two heuristics--one which it is hypothesized is better in the long-term, and the other is hypothesized to be better in the short-term. The question then becomes one of how we combine the two heuristics into a single one in a natural way.

Essentially the agent is interested in only one thing--the value of the action that it ends up performing. The two heuristics that we have measure different things. The action-based heuristic measures the immediate value of the improved action gained by performing a test, while the information-theoretic heuristic, which makes the best use of time for long deadlines, we are hypothesizing is a good predictor of the value of future actions (until we verify this experimentally later on this will be only a hypothesis). Whenever a test is performed, there is a certain potential value of that test that would be reached only if the test has zero cost and finds actions of value 1.0 for all faults. After the test is performed some value will actually be gained--the value of the improved actions obtained by performing the test. And some potential value will be lost--the difference between a perfect action and the actual action obtained should the deadline be reached while the test is being performed. This is illustrated in Figure 3, which shows these concepts applied to the situation shown in Figure 2. The initial value of the best action is 0.25, which is represented by the black region. Because potentially there could be a test with zero cost that identified an optimal action, all the rest of the diagram represents the potential future value at time zero. When the first test is performed, taking time 2, it identifies an action, A1, which has value 0.5. Now the gray area is essentially lost value--if the deadline occurs between time 0 and time 2, the best action will have value 0.25, and there is now nothing the agent can do about it. On the other hand, the striped area represents the current value at time 2, in addition to the black area. Now, only the white area represents potential future value.

It is important to realize that the term potential future value being referred to here represents the potential future value given that the hierarchy has only been expanded down to a particular node. It is therefore a concept that has meaning only at reactive planning time, not when the agent is executing the plan. Nevertheless, at hierarchy construction time, the reactive planner has available to it only information about the state of the hierarchy above the node it is trying to expand, so it is appropriate that its heuristic be based only on this information.

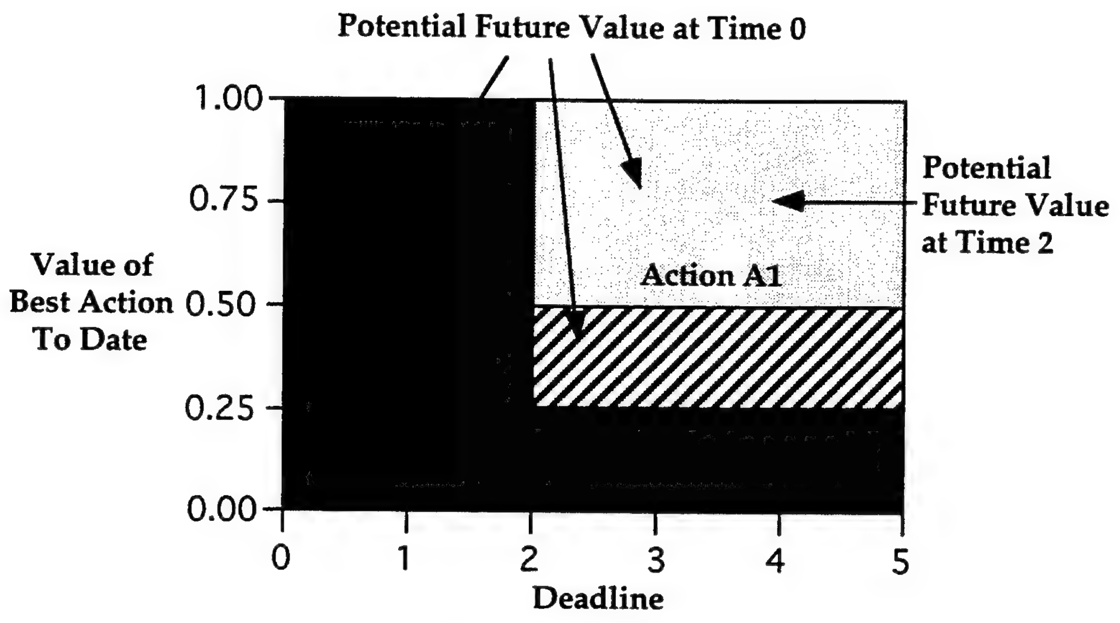


Figure 3:
Potential vs. Actual Values of Action

The heuristic that we propose using is to calculate both the value that will be realized by performing a particular test, and the percentage of the potential future value that the reactive planner expects will be realized. Furthermore, the information-theoretic heuristic is used to estimate what percentage of potential future value will actually be realized. In other words, we use the following heuristic:

$$A + PI$$

where A is the action-based heuristic, P is the potential future value, and I is the information-theoretic heuristic. Here A is the action-based heuristic taking into account deadline distributions described earlier:

$$A = \frac{V_{Gd}(t,F_n,E_n)+V_{GI}(F_n,E_n,C(t))-V_{Gd}(F_n,E_n)}{C(t)}$$

I is the information-theoretic heuristic given as a percentage. The information-theoretic heuristic is originally expressed as a negative number representing the number of bits of information needed to completely diagnose a fault. Therefore by comparing the number of bits of information still needed at the parent node to the number gained in moving to a child node, we can express the information-theoretic heuristic as a percentage. Specifically, this works as follows:

$$I = 1 - \frac{\sum_{o \in t} P(o)I(o)}{\sum_{f \in F_n} \frac{P(f)}{P(F_n)} \log \left(\frac{P(f)}{P(F_n)}\right)}$$

Finally, the reactive planner needs to be able to compute P, the potential future value after performing the test. It is given by the following formula:

$$P = V_p(F_n,E_n + C(t))$$

where

$$V_p(F_n,E) = \frac{\sum_{f \in F_n} P(f)\, V_p(f,E)}{\sum_{f \in F_n} P(f)}$$

and

$$V_p(f,E) = \int_E^{\infty} D_f(t)\,dt$$

Our hypothesis for the performance of hierarchies constructed using this heuristic is that they will perform roughly as well as the maximum performance of either the information-theoretic or the action-based deadline distribution heuristic.

Section 3.3 -- Multiple Tests at a Time

We now seek to relax the first and second assumptions stated at the beginning of this chapter. Specifically, the planner now takes the physical cost of tests into account in deciding when to perform a test. The idea is that the reactive agent may wish to perform a test in advance of its really being needed so as to avoid the delay in acting that would be involved if it had to wait for the test's outcome later on. That is, when the agent reaches a particular node in the hierarchy in its diagnosis process, it may decide to perform not only the test associated with that node but also the tests associated with one or more of its children or other descendants. The process by which it does so is known as *test promotion* because it involves "promoting" a test so that it is performed earlier than it is really needed. The hierarchy is first structured using the deadline distribution heuristics described in the last section (deadline distribution is essential to take into account in deciding whether to do test promotion). Then the test promotion process is done. Test promotion involves deciding whether the cost in performing a test earlier than necessary is greater than the benefit gained by performing it early. The basic algorithm is fairly simple:

Algorithm 3 -- Test Promotion

Step	Description of Step
1	Structure the hierarchy using the algorithm given above.
2	Set the current node to the root node in the hierarchy.
3	First recursively do test promotion by executing steps 2 to 4 with the current node being set to each of the children of the current node.
4	Then for each child in turn of the current node, determine whether the performance profile will improve or be worse if the tests for that child are moved to the current node. If the profile will improve, move the tests up.

In order to execute step 4, we need to expand the notion of a node to be a 6-tuple $<L, t, a, F_n, E_n, Q_n>$ where E_n is the amount of time that will have elapsed in executing the reactive plan in getting to this node, and Q_n is a set of ordered pairs of tests and times, representing the set of tests that will have been performed up to and including the time this node is reached in the reactive plan. The values of the E_n may be computed top-down as follows:

$$E_R = 0 \text{ for the root node } R$$

If a node $<L, t, a, F_n, E_n, Q_n>$ has a child $<L_i, t_i, a_i, F_{n_i}, E_{n_i}, Q_{n_i}>$ then

$$E_{n_i} = \max(E_n, Q_n(t) + C(t))$$

Initially the Q_n's are set as follows:

$$Q_{n_i} = Q_n \cup \{<t_i, E_{n_i}>\}$$

At all times the Q_n's must obey the following property: if for some test t_j there is an E such that $<t_j,E> \in Q_{n_i}$ but no E such that $<t_j,E> \in Q_n$, then $E = E_{n_i}$.

Intuitively, this means that when the test promotion process begins in the hierarchy structuring process, all tests are performed only when the node requiring the outcome to the test is reached. What is at issue in deciding to "promote" the test(s) associated with node n_i to the node n is whether the tests that are performed when node n_i is reached would be better performed earlier--when node n is reached. If the decision is "yes" then the following substitution will take place:

- For each $<t_j,E_j>$ in Q_{n_i} if there is no $<t_j,E> \in Q_n$ for any E, then do:
$$Q_n \leftarrow Q_n \cup \{<t_j,E_n>\}$$
- $Q_{n_i} \leftarrow Q_n$
- The Q and E values for all nodes below n_i in the hierarchy will need to be modified to be consistent with the above properties.

In order to decide whether to promote a test, we need to compare cost and performance profiles for the node n for the case where we promote the test versus the case where we do not promote the test. The performance profile will be the integral of a step function defined for time points greater than or equal to E_n. It is equal to the weighted average of the performance profiles for individual faults:

$$V(n) = \sum_{f \in F_n} P(f) \int_{E_n}^{\infty} D_f(t)\, V(f,n,t)\, dt$$

The performance profile for each fault may be computed recursively. Suppose that there are k outcomes of the test t, $\{o_1, o_2, ..., o_k\}$, such that $f \in o_i$. (Actually, because in the current section the assumption is that test outcomes are disjoint, for the moment $k = 1$, but the more general analysis is provided here to avoid needless repetition in the next section.) Then if the corresponding children nodes are $<L_i, t_i, a_i, F_{n_i}, E_{n_i}, Q_{n_i}>$ for $1 \le i \le k$, the recursive definition for the performance profiles is:

$$V(f,n,t) = V(f,a) \text{ for } E_n \le t < E_{n_1}$$

$$V(f,n,t) = \frac{1}{k}\sum_{i=1}^{k} V(f,n_i,t) \text{ for } t \ge E_{n_1}$$

and for leaf nodes:

$$V(f,n,t) = V(f,a) \text{ for } t \geq E_n$$

In order to determine whether to promote a test, the reactive planner also needs to compute cost profiles. In so doing, the agent will for the first time take into account the physical costs of tests defined in the last chapter. Cost profiles are defined recursively in a similar way to performance profiles:

$$S(n) = \sum_{f \in F_n} P(f) \int_{E_n}^{\infty} D_f(t) S(f,n,t)\, dt$$

However the basic definition of cost differs from that of performance. The cost at any point in time refers to the sum of the physical costs of all tests performed up to that point in time. Formally,

$$S(f,n,t) = \sum_{x \in Q_n} S(\pi_1(x)) \text{ for } E_n \leq t < E_{n_1}$$

where $\pi_1(x)$ denotes the first (test) element of x. Also:

$$S(f,n,t) = \frac{1}{k} \sum_{i=1}^{k} S(f,n_i,t) \text{ for } t \geq E_{n_1}$$

and for leaf nodes:

$$S(f,n,t) = \sum_{x \in Q_n} S(\pi_1(x)) \text{ for } t \geq E_n$$

Having completed this analysis of cost and value profile, the reactive planner will promote tests if the value-cost difference $V(n)$-$S(n)$ is greater after test promotion than before.

One final point which needs to be raised is that if this difference is negative, the cost of further diagnosis from this node exceeds the benefit, and the agent, when reaching this node in the hierarchy, should call a halt to performing further tests and immediately perform the action associated with the current node regardless of when the deadline is.

This algorithm for test promotion is essentially a hill-climbing algorithm. Thus, we know that the final reactive plan that results from the final hierarchy will be better, on average, than the one we start with. However, it cannot be guaranteed that there may not be still better reactive plans that lie hidden in the search space. Nevertheless, this algorithm seems likely to promote the most important tests: if a test is quite low cost but delaying performing it would be very expensive, then this algorithm will detect this and do so. Thus, the proposed algorithm seems to represent a tradeoff between the need to produce something computationally tractable and something that would produce the optimal hierarchy.

Section 3.4 -- Non-Disjoint Test Outcomes

The assumption has been made to date that the outcomes for a test are always disjoint, or alternatively that each test is a partition of the set of faults or that any test has exactly one possible outcome for a given fault. In this section we examine what happens when this assumption is relaxed to the weaker assumption that a test may have multiple outcomes consistent with a given fault, but those outcomes will all be equally likely in the presence of that fault.

The primary difference that this imposes is that now it is possible for a node in the hierarchy (which corresponds to a set of faults) to have multiple parents, whereas before a node could have only one parent. It will therefore be necessary for the agent, in executing the reactive plan, to keep track of the set of tests that have been executed in reaching a particular node in the hierarchy. Furthermore, each node will have associated with it a set of tests, instead of just a single test, and when the node is reached, the highest ranking test on that list that has not previously been executed will be performed. Each node will have a number of sets of children, one set corresponding to each of the tests that can be performed at this node in the hierarchy.

Structuring the hierarchy will be proceed similarly to before. The only difference will be that when a child node is potentially to be created, the reactive planner will look at the list of existing nodes and determine whether a node already exists with the same set of faults. If so, no new child node will be created, but rather a pointer will be created to the existing child node. The planner will then determine whether there is a test to be performed at the child node consistent with this trajectory through the hierarchy. If there is not, further expansion of the child node will be necessary; if there is, then no further expansion is necessary. Step 6 in the hierarchy structuring algorithm will change to determine the best test among those that have not already been performed in reaching this node in the hierarchy.

Formally, the only difference is that the 6-tuple that defines each node now becomes:

$$<<L_1,\ldots,L_k>,<t_1,\ldots,t_k>,a,F_n,<E_{n1},\ldots,E_{nk}>,<Q_{n1},\ldots,Q_{nk}>>$$

so that essentially each node now has a set of children sets instead of just one.

One important assumption has been made in order to reduce the possibility of a combinatorial explosion in expanding the search space. This regards the prior probabilities of lower nodes in the hierarchies. Because of the differing numbers of competing outcomes that different faults may have for a given test, the prior probabilities of faults in lower nodes in the hierarchy may not be in the same proportion as the original prior probabilities of the faults. However, the above approach assumes that they are. The reason this simplifying assumption is made is that were it not for the pruning of the search space that occurs when different branches lead to the same node, the search would quickly become computationally intractable. If we needed to have a different node for every possible combination of prior probabilities, much less pruning would go on. It is not clear to what degree this assumption will degrade performance. It seems likely that if the original probabilities differed by an order of magnitude, the assumption would not affect structuring and performance much; if they were a lot closer, then it would make more of a difference.

Section 3.5 -- Multiple Faults

There are essentially four possible approaches that can be taken to handling the multiple fault problem. The first is that when we assume that tests may have non-disjoint outcomes, it becomes entirely plausible that certain leaf nodes in the hierarchy will include multiple faults. In this case, actions can be designed for that combination of faults rather than for individual faults. This possibility for diagnosing multiple faults occurs naturally as a result of the work described up to this point, and does not require any further work on the part of the reactive planner.

For example, a particular test may have three outcomes:

{F1,F2,{F1,F2}}

meaning that either fault 1 is present (only), fault 2 is present (only) or both faults are present. If the third outcome to this test occurs, and there are no other tests that distinguish between these two faults, then the diagnosis of the pair of faults F1, F2 can be made.

The second approach is to specifically encode in the knowledge base information about certain pairs or small sets of faults that are known to have interesting interactions that the agent should know about. Obviously, it would not be possible to do this with all possible combinations of faults, but it should be possible for the most likely combinations of faults. These combinations of faults will then become, for the purposes of both the reactive planner and the reactive agent, single faults. Only the designer of the knowledge base will know that they are actually combinations of faults. This approach seems reasonable in view of the fact that it is not the goal of the reactive agent to encode responses to all possible contingencies, but rather to be prepared to respond to the most important and urgent faults and combinations of faults in a timely fashion.

A more sophisticated version of this approach is for the reactive planner to explicitly know the component faults of these important combinations of faults. This would allow the reactive planner to use the knowledge about the value of an action for a particular fault in the presence of other faults (which may differ from the value when the single fault occurs alone). However, the reactive planner would still, for the purposes of executing the hierarchy structuring algorithm, represent this combination of faults as a single fault.

The third approach is to allow the execution of any test to return multiple outcomes, so that a single search through the hierarchy by the reactive agent may lead to diagnosing multiple faults. This approach requires that the agent be capable of making a control decision as to which branch of a hierarchy to explore next in the event that it discovers the presence of multiple faults. The most reasonable way of doing this appears to be to use the same heuristic that is used to structure the hierarchy to also decide which branch of the hierarchy to explore next in invoking the reactive plan.

The final approach is applicable in the case of dynamic replanning, and requires that the knowledge about values of actions in the presence of multiple faults be available. This approach is simply to notice that when doing replanning, knowledge about certain occurring faults may already be available, and the fact that these faults are present can be used to structure the hierarchy.

These different approaches to handling multiple faults are not intended to be mutually exclusive. Indeed, all four approaches could be used by the same agent. The multiple fault problem is a difficult one and by providing several ways of dealing with multiple faults, the agent has the best chance of being able to deal with them effectively.

Section 3.6 -- Related Work

The idea of varying the heuristic in structuring decision trees is not new. What is new about this application of decision trees is that they are being used not merely to do classification, but also to provide meaningful responses in the interim when a complete classification is impossible. Since the heuristics used here are tailored to this application, they have not been previously used in the literature. It is because of this substantially different application that we prefer a different term, action-based hierarchies, for our decision trees.

There is a fairly extensive body of literature devoted to exploring different heuristics for decision tree structuring. An information theoretic heuristic is used by various researchers: by Cheng et.al. in GID3 [CH88], by Fayyad in GID3* [FA91], by Quinlan in C4 [QU90], by Breiman et.al. in CART [BR84], and by Clark and Niblett in CN2 [CN89].. Fayyad and Irani introduce a class separation approach (C-SEP in [FA92]) where the heuristic measures not the information content of a test but the degree to which it separates faults into the different classes. This presupposes, however, that the goal of using the decision tree is to diagnose down to the level of an individual class, not an individual fault. It is unclear, therefore, how this approach could be applied to the current problem without some delineation of classes that would in turn require applying the action-based heuristic first. GID3 and GID3* differ from ID3 (the original information-theoretic approach of Quinlan) in that only some of the outcomes of a test are used in branching. Many other approaches have also been taken in the literature.

A novel approach to structuring decision trees, *oblique decision trees*, is introduced by Murthy et.al. [MU93]. The idea here is that their algorithm, OC1, determines the optimal oblique hyperplane with which to split the set of data points. The equivalent problem for us would be to design our own tests, rather than using a pre-defined set of tests from a given domain. This would be an interesting topic for further research, but is beyond the scope of the current paper.

Chapter 4
Complexity Analysis

Although the main focus of the research described in this thesis is to produce reactive plans that can be executed with a minimum of computational resources, an analysis of the complexity of the reactive planner is also necessary because if it is prohibitive to use the reactive planner (if its complexity were exponential in the size of one of its inputs, for example) then the approach would be less attractive. This analysis is also important in the event that any dynamic replanning is done because it will shed light on how expensive that task is, and also on the number of faults that can reasonably be handled in a reactive plan.

There is also a space complexity issue involved. Potentially, there is one node in the hierarchy for every subset of the whole set of faults. If this exponential potential ever came close to being realized, then there would not be sufficient space to represent the reactive plan. Also with regard to space complexity, one of the assumptions made in structuring the hierarchy was designed to reduce the space complexity of the reactive plan. This assumption was that the prior probabilities of the faults at any node in the hierarchy are in the same proportion as they are at the root node of the hierarchy. The reason for this simplifying assumption was to eliminate the need to have separate nodes for the same set of faults with differing probabilities; if this were allowed, then the potential to use up a lot of space in representing the different probabilities for the same set of faults would be great.

There is no space complexity issue for the reactive planner independent of that of the reactive plan generated. That is, the reactive planner does not use significant space in generating the reactive plan beyond the space required to represent the plan itself, so if there is sufficient space to represent the reactive plan, the reactive planner will have no space difficulties.

The time complexity of the reactive plan is, by design, trivial. The only tasks in executing the reactive plan are to recommend tests when the appropriate node in the hierarchy is reached, and in the case of multiple faults, to handle the control problem of which path through the hierarchy to examine first. The former case is merely a table lookup, and the latter involves comparing the two (or more) heuristics for the different parts of the hierarchy, which is also trivial. This shows that although in theory it is necessary to analyze the complexity of both the reactive plan generated and the reactive planner, all that really needs to be learned can be determined by looking at the complexity of the reactive planner.

In general, the biggest difficulty in analyzing the complexity of the approach described in chapter 3 comes in estimating the number of nodes in the generated hierarchy; this issue affects both the time and space complexity of the approach. The number of nodes can be reduced by coalescing repeated nodes (different nodes with the same set of faults, etc.) into single nodes, producing a *decision graph* as opposed to a decision tree. This problem is easy in the case where the outcomes of tests are disjoint, because then there will be one terminal node for each fault, and fewer non-terminal nodes than there are faults. When the set of test outcomes is non-disjoint, there is as was mentioned above potentially one node in the hierarchy for each subset of the set of faults, which would be a prohibitively high space cost. However, in practice it does not work out nearly so badly, so the formal analysis will be augmented by actual examples from a real domain--surgical intensive care unit patient monitoring. A description of this domain may be found in chapter 7.

A further simplification in the complexity of running the hierarchy structuring algorithm occurs because it is not in general necessary to have an explicit representation of the value of each action for each fault in order to run the algorithm. Specifically, it is possible to represent each action as having value only for a few faults, and having a fixed cost which does not vary depending upon the fault. Computationally, this will greatly simplify the algorithm. It should also be noted that the information-theoretic approach does offer an advantage over the action-based approach in terms of the complexity of structuring the hierarchy, largely because it takes into account only the prior probabilities of the faults, whereas the other heuristics take into account a variety of other variables. In the remainder of this chapter we will examine in more detail how significant this advantage is.

Section 4.1 -- Basic Problem

We first give the complexity for the solution to the basic problem described in section 3.1. Time complexity will be defined in terms of the following:

- the number of faults, given by **f**
- the number of actions, given by **a**
- the number of tests, given by **t**
- the mean number of outcomes to a test, given by **o**

We first define the time complexity of expanding a particular node **n** which has **n** faults associated with it. We first need to compute, for each test, the intersection of the outcome sets with the fault set associated with the node--this will take $O(\mathbf{on})$ time on average. The next step is the time to determine the best action for each potential child set, which will be $O(\mathbf{an})$ on average, so the time so far is $O((\mathbf{a}+\mathbf{o})\mathbf{n})$. These represent the time-dominant steps in the algorithm--but they must be repeated once for every test, for a total of $O(\mathbf{t}(\mathbf{a}+\mathbf{o})\mathbf{n})$ time. So this gives the time complexity to expand a given node. There are $O(\mathbf{f})$ nodes in the hierarchy, and the average node has at most $\mathbf{f}/2$ faults associated with it. This gives a total time complexity of

$$O(\mathbf{t}(\mathbf{a}+\mathbf{o})\mathbf{f}^2)$$

Since there are $O(\mathbf{f})$ nodes in the hierarchy with $\mathbf{f}/2$ faults associated with each on average, the space complexity will be

$$O(\mathbf{f}^2)$$

However, the time complexity given above assumes that each action has an interesting non-zero value for each fault. If value of actions are represented such that the value of each action is non-zero for only a small percentage **k** of faults, then the time complexity is greatly reduced to

$$O(\mathbf{kt}(\mathbf{a}+\mathbf{o})\mathbf{f}^2)$$

Now, with regard to time complexity of the information-theoretic structuring heuristic, the main saving will be that one does not have to compute the value of each action for each fault, so it is reduced to

$$O(\mathbf{tof}^2)$$

In principle this is considerably better than the action-based case, so this would be a point in favor of using information-theoretic hierarchies. Note, however, that if the percentage of actions with positive value for a given fault is small, something that one would expect to be true in general in most domains, then **k** will be small and the time complexity for the action-based case will be roughly equal to that of the information-theoretic case--no similar saving can be realized for the information-theoretic case. Thus, in practice it seems that there will be little savings in time complexity by using the information-theoretic hierarchy.

It also needs to be noted that this is a worst-case scenario where each test only succeeds in ruling out or identifying one fault. The best-case scenario, which is likely to be closer to average, gives a time complexity of

$$O(\mathbf{kt(a+o)f} \log_2 \mathbf{f})$$

for the action-based case and

$$O(\mathbf{tof} \log_2 \mathbf{f})$$

for the information-theoretic case. These complexity values use a hierarchy depth of $\log_2 \mathbf{f}$ which is the best possible case given the assumptions made.

Section 4.2 -- Deadline Distributions

Addressing the complexity of the solution to the deadline distribution problem first requires that one evaluate the complexity of actually computing the integrals provided in the approach to this problem. The assumption that will be made is that the integrals can be computed in constant time. This assumption is made for two reasons: first, many distributions will not actually be given as complex functions to be integrated but rather as simple step functions. For example, it might be given that a particular fault has a deadline between 20 and 30 minutes or with a mean of 25 minutes. In the former case, a uniform distribution can be assumed, and in the latter case, a normal or exponential distribution can be assumed. It is not that these assumed distributions are likely to be perfectly accurate, but rather that there is unlikely to be any precise formula for giving a perfectly accurate distribution, so approximations must be used. In any event, these approximations have known easy methods for computing the integral which are constant time.

Second, even in the event that a complex function requiring Simpson's rule to integrate, an approximate Simpson's rule evaluation using the value of the function at perhaps 16 points should be sufficiently accurate for the purposes of this problem, and will require only constant time to compute.

Under this assumption, the complexities for the deadline distribution case will be exactly the same as when the deadline distributions are not taken into account--for time, it is

$$O(\mathbf{kt(a+o)f} \log_2 \mathbf{f})$$

and for space it is

$$O(\mathbf{f} \log_2 \mathbf{f})$$

These are average complexities. Of course, the actual time taken to structure the hierarchy will be longer for the deadline distribution case; it is just that it has the same order of complexity because it differs by a constant factor.

Section 4.3 -- Test Promotion

Because the first step in the test promotion algorithm is to structure the hierarchy using the regular deadline distribution approach, the complexity for that portion will be the same as above. It is therefore the second part of the algorithm, where actual promotion of tests is done, that we are concerned with here. In order to determine this we look at the complexity of doing test promotion for a given node in the hierarchy. The two steps in test promotion are first to ensure that accurate timing information is known for all lower nodes in the hierarchy. If the node is $\mathbf{n}$ with an associated fault set of $\mathbf{n}$ faults, then this process will take time proportional to the number of faults, hence it is $O(\mathbf{n})$. The second step is to determine the cost and performance profiles for each fault in the fault set, and then average them. This will be proportional in cost to the number of faults multiplied by the average depth of the hierarchy below node $\mathbf{n}$, and hence it is $O(\mathbf{n} \log_2 \mathbf{n})$ (given the depth of the hierarchy below node $\mathbf{n}$ is $\log_2 \mathbf{n}$). Thus the time complexity of doing test promotion for a single node is $O(\mathbf{n} \log_2 \mathbf{n})$. To do it for the entire hierarchy the time complexity will be

$$O(\mathbf{f}^2 \log_2 \mathbf{f})$$

Space complexity will not change by virtue of doing test promotion. It should be noted that depending upon the relative sizes of $\mathbf{f}$ and $\mathbf{kt}(\mathbf{a}+\mathbf{o})$ it may be either the test promotion phase or the regular structuring phase of the process that is dominant in terms of the time consumed. However, in general one would expect that the latter term would be larger, and hence the test promotion phase of the algorithm is cheap compared to the rest of the algorithm. In other words, there is little reason not to do test promotion if one is going to do the rest of the algorithm.

However, two caveats must be noted here. The first is that there is little obvious that one can see to do to reduce the time complexity of the test promotion phase. For the structuring phase, providing explicit representations only for those actions with positive values for particular faults provides, in practice, significant savings in time complexity. For the promotion phase, the above complexity seems to be about the best one can do. It should also be noted that although test promotion guarantees a better hierarchy than the one started out with, it does not guarantee the best possible hierarchy. That is for two reasons: test promotion always moves tests up, not down, and there could be cases where moving a test back down provides even better performance; and tests are moved up or not as blocks, not as individual tests. To produce an algorithm that used test promotion to produce the best possible hierarchy would likely be NP-complete.

Section 4.4 -- Non-disjoint Test Outcomes

As outlined above, the case where test outcomes are non-disjoint is awkward because it is difficult to estimate the actual number of nodes in the hierarchy. First off, non-disjointness increases the time complexity of finding the best action for each child node set to O(**aon**) from the previous value of O((**a**+**o**)**n**). As before, this must be repeated once for each test, giving a complexity for expanding each node of O(**taon**). If the total number of nodes in the hierarchy is **s** then the total time complexity of doing hierarchy structuring is O(**staof**).

How does one measure the number of nodes in the hierarchy? One way of doing it is to estimate the discriminating power of particular tests. If the average outcome of a test reduces the fault set to a certain proportion **r** of its original size, then it is possible to estimate the total number of nodes in the hierarchy to be $\mathbf{s} = \mathbf{f}^{-\log_{\mathbf{k}}\mathbf{o}}$, which therefore results in a total time complexity of

$$O(\mathbf{taof}^{1-\log_{\mathbf{k}}(\mathbf{o})})$$

which is polynomial in the number of faults **f**.

However in practice it is not so bad. In the medical intensive care unit domain described in chapter 7, the problem has, approximately,

$$\mathbf{t}=45, \mathbf{a}=58, \mathbf{f}=58, \mathbf{o}=3, \mathbf{k}=0.9$$

The expected number of nodes would therefore be 2.44×10^{18} if the above formula were to be believed! The actual number of nodes in the generated hierarchy is a quite manageable 105 nodes.

This enormous (and advantageous) difference between the theoretically predicted and actual number of nodes is caused primarily because the theory assumes essentially no structure on the problem, whereas when an actual problem is presented, it is quite well structured and hence what would have been a combinatorial explosion of nodes becomes a quite reasonable hierarchy. The experience with this particular problem would appear to indicate that the number of faults represents a good order of magnitude estimate of the number of nodes in the hierarchy, even though theory predicts something much worse.

It is interesting to note that the hierarchy structuring heuristic appears to have some bearing on the number of nodes in the generated hierarchy. When the information-theoretic heuristic is used, the number of nodes in the hierarchy is 376. When the action-based heuristic is used, the number of nodes drops to 276 even when costs of tests are not taken into account; when costs of tests are taken into account the number drops to the above-mentioned 105. It thus seems that the action-based heuristic, although not designed to reduce the number of nodes in the hierarchy, has this effect in this particular domain, and furthermore that considering the costs of tests is an important component in keeping the number of nodes down. It would be interesting to explore the reasons why this is the case.

When deadline distributions are taken into account, there should be no change in time complexity of the structuring algorithm, for the same reasons as described in section 4.2.

If test promotion is done, then as before there are two steps to this phase of the algorithm: determine accurate timing information for lower nodes in the hierarchy and then determine cost and performance profiles for each fault. For a given node **n** with **n** nodes below it in the hierarchy (this is a different definition of **n** than that used previously) the time taken to determine timing information will be proportional to **n** ($O(\mathbf{n})$). For determining cost and performance profiles, the time complexity will be the number of faults multiplied by the number of nodes: $O(\mathbf{nf})$.

This gives time complexity for test promotion of one node in the hierarchy. For the average node in the hierarchy, the number of nodes below it will be roughly equal to the square root of the total number of nodes in the hierarchy, so if **s** is the total number of nodes in the hierarchy, then the total time complexity is given by

$$O(s^{3/2}f)$$

In this case, test promotion may be the dominant part of the algorithm (note that in the non-disjoint case, it usually was not the dominant part). The reason is that the number of nodes (which is the large uncertain factor in this process) has a higher exponent for the test promotion part of the algorithm than for the structuring part of the algorithm. Since the number of nodes is potentially quite large, if this potential is realized, it may be wise not to proceed with the test promotion phase of the algorithm.

To summarize what we have learned about complexity thus far in this chapter: for the non-disjoint case both time and space complexity are quite reasonable, especially if the fact that values of actions need be represented explicitly for only a few faults is taken into account. However, when test outcomes become non-disjoint, the number of nodes has the potential to increase dramatically. In practice, it has been observed that at least one problem domain is sufficiently well-structured that this unhappy outcome does not occur. If the number of nodes is large but still manageable, then it may be unwise to perform the test promotion algorithm, since that part of the algorithm may put things outside the range of being computationally tractable.

Section 4.5 -- Multiple Faults

Chapter 3 outlined four possible approaches to the multiple fault problem. Two of these approaches--incorporating knowledge about co-occurring problems into the hierarchy itself and having the reactive plan execute tests for different areas of the hierarchy at the same time--do not bear directly on the hierarchy structuring problem. Performing searches of multiple areas of the hierarchy occurs at run time after structuring is complete, and hence has nothing whatever to do with complexity. Putting knowledge about co-occurring faults into the hierarchy requires knowledge about combinations of faults to be available to the structuring algorithm and hence, for complexity purposes, is a sub-case of the fourth approach described below. Another approach, using the non-disjointness of test outcomes as a means of diagnosing multiple faults, arises naturally from the hierarchy structuring algorithm and requires no further analysis.

The fourth approach is for the hierarchy structuring algorithm to know about interesting combinations of faults and to treat these combinations as single faults in structuring the hierarchy. There were actually two different ways the hierarchy structuring algorithm could handle this. One is for that algorithm to simply deal with the combinations of faults as though they were single faults and not have any knowledge at all that they are really combinations of faults. In this case, the effect on complexity is obvious: it simply increases the number of faults and hence changes all complexity values dependent on the number of faults, but there are no other changes in complexity evident.

The more sophisticated approach was for the hierarchy structuring algorithm to know about particular combinations of faults as combinations of faults (not as single faults) and also have knowledge about the values of actions in the presence of other faults (which may vary from the value if no other faults are present). In this case, suppose that for any given fault-action pair, there are **c** different combinations of other faults, with an average of **l** faults each, that the hierarchy structuring algorithm knows about and knows how they affect the value of the action for the fault. For each such combination, the structuring algorithm must determine first whether it is a subset of the fault set of the current node: this takes on average O(**l**) time. Then the structuring algorithm must determine which of the combinations of faults are maximal subsets; this can take up to $O(c^2l)$ time; the total time taken to determine the value of the action for the fault is thus $O(c^2l)$. The effect on the time complexity of the entire algorithm will thus be as though the number of actions was increased from **a** to ac^2l.

This amounts to a greater time complexity price to pay in terms of structuring the hierarchy for the more sophisticated approach, which is what one would expect. The correct approach would be to try to keep **c** and **l** as low as possible. In fact, if the added information is known only about a few small combinations of faults, then these values will be only slightly greater than 1, and hence the effect on time complexity of the structuring algorithm will be minimal.

As mentioned previously, if knowledge about certain faults being present is already known to the hierarchy structuring algorithm, then this information can be used to help structure the hierarchy. This will not affect complexity in any significant way; it will simply reduce the number of faults that must be considered slightly, and will force examination of all other faults under the assumption that certain faults are present.

There is nothing stopping one from using a combination of the two approaches, of course, providing detailed information about the most common combinations of faults and providing the less detailed information about the less common, but more numerous, combinations. Indeed, it seems that the greatest cost to be paid in terms of knowing about multiple faults is in the knowledge acquisition phase, not the hierarchy structuring algorithm.

Chapter 5
Formal Results

Ideally, one would be able to prove formally that the approach presented in chapter 3 represents the best possible solution to the problem given in chapter 2. If this could be done, the thesis could immediately be concluded without the need for any experiments or examples! Although it turns out this cannot be done, in this chapter three formal results will be presented. The first result shows that under certain well-defined assumptions, the action-based hierarchy provides the best performance in the strong sense that no matter what the value of the deadline and no matter what other structuring heuristic might be used, the action-based hierarchy will perform at least as well. The second result gives similar results in the case where the hierarchy is very small. The third result shows that when test promotion is done, the performance of the resulting hierarchy will be at least as good as the hierarchy that it replaces.

Section 5.1 -- Optimal Action-Based Hierarchies

The assumptions that are required to make the claim that action-based hierarchies provide optimal performance are the following:

- All the assumptions made for the most basic version of the problem described in chapter 3 must be made.
- In addition, one of the following two conditions must hold:
    - The temporal costs of all tests must be equal.
    - There must be no time wastage at the deadline: that is, the deadline does not occur during the middle of performing a test but rather immediatetly after a test result comes back.
- The really strong assumption that must be made is that the values of tests are additive for each fault. The value of a group of tests for a fault is the amount by which the value of the best action available for that fault improves after performing the group of tests versus what its value is before performing the group of tests. Additivity means that the value of any group of tests is the sum of the values of performing each test.

Under this set of assumptions, the value of the best action can be taken to improve continuously as a linear function. This may not actually be true (the value improves as a step function) but the second assumption means that it can be taken to be true. If there is no time wastage at the deadline, then the improvement in the value of the best action between one step down the hierarchy and the next can be taken to occur linearly over that interval, because it is not really important exactly how it occurs. If the temporal costs of all tests are equal, then the performance of the hierarchy can be taken to be the same as if there were no time wastage at the deadline.

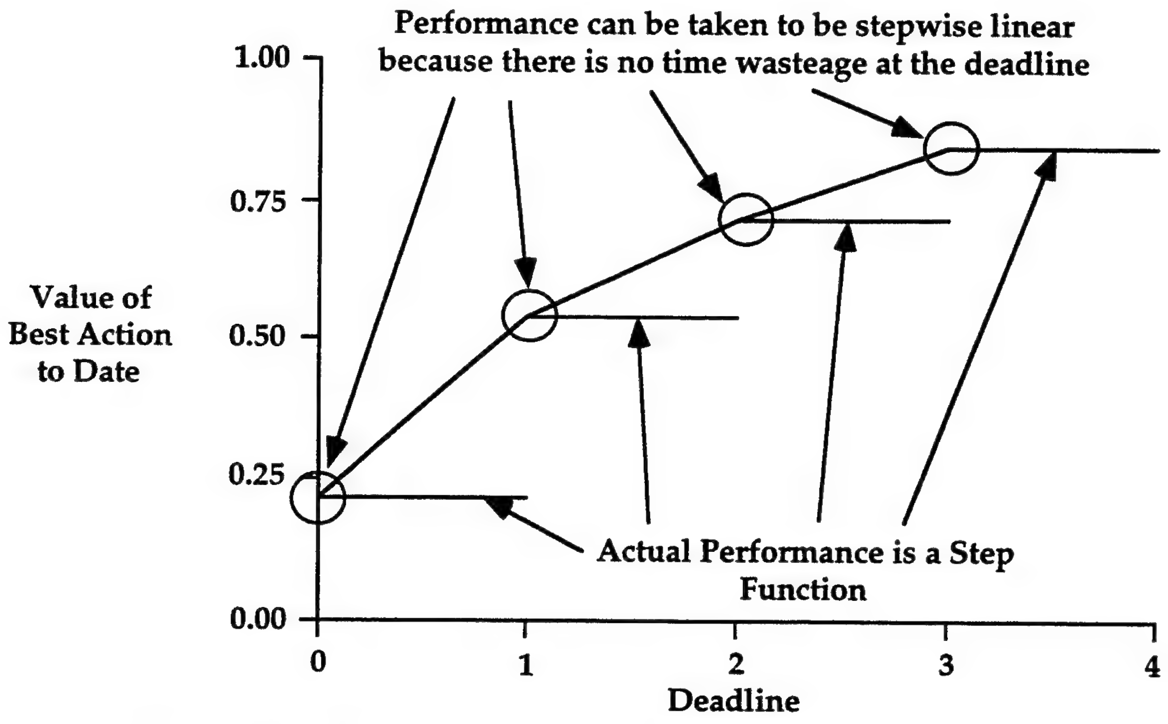


Figure 1: Under strict assumptions actual performance may be approximated by a stepwise linear function

This is illustrated in Figure 1. The actual performance of the hierarchy is shown by the step function pointed out by the arrows in the bottom half of the diagram. Because of the assumption that there is no time wastage at the deadline, however, the only relevant points on this graph are those points where a refinement of the diagnosis has been made. These points are illustrated with the small circles. Hence under these assumptions, it is perfectly reasonable to also assume that the actual performance of the hierarchy is given by the piecewise linear function pointed out by the arrows in the top half of the diagram. And when the cost of all tests are equal (as they are in this diagram) if we can show that one hierarchy outperforms another at all the circled points, then it does so at the intermediate points as well.

What does being able to make this additional assumption buy us? The slopes of each segment of the piecewise linear function represent the gain per unit time that performing the corresponding test buys the agent. Because the value of performing tests is additive, this slope will be the same no matter when in the reactive plan a given test is performed. Hence it behooves the agent to perform the highest slope test first, then the second highest, and so on. This amounts to just the action-based hierarchy structuring heuristic: so under this set of assumptions the performance of such a hierarchy is necessarily optimal at all relevant time points.

It is clear that this set of assumptions is far too strong to hold in most real-world domains: the question is more whether they hold in any domain at all. One domain in which it is possible that they come close to holding might be managing a stock portfolio in a volatile market, consisting of a number of stocks each of which may change in price independently. In such a domain, a test would consist of analyzing the recent performance of a stock to determine whether it is necessary to take immediate action to sell or buy shares in it, and which type of action is appropriate. Additivity of tests comes about in this domain because the expected value of each stock transaction is essentially independent of the other transactions. Furthermore, if an approximately equal amount of data is available for each stock, it may be reasonable to assume that the cost of performing each test is equal.

In such a domain, the action-based hierarchy approach is guaranteed to be optimal. However, this domain is not an interesting application of this technology because it essentially consists of a set of decoupled problems that the agent is working on. To put these problems together in an action-based hierarchy would be both space wasteful and unnecessary in view of the fact that the existing technology of Dean and Boddy's deliberation protocols for anytime algorithms can handle this type of problem.

It is not clear that any domain in which this set of assumptions holds does not have this property. The conclusion from this is simply that there is a strict limit to how far one can go formally with this approach. In the next section we will consider a different set of assumptions under which it can also be proven that action-based hierarchies are optimal, but which can represent an interesting application of this technology.

Section 5.2 -- Very Small Hierarchies

A similar claim can be made--that action-based hierarchies provide better performance than any possible competitor--under the following conditions which mandate a very small hierarchy:

- All the assumptions made for the most basic version of the problem described in chapter 3 must be made.
- The temporal costs of all tests must be equal.
- At most two tests are required to complete the diagnosis process (hence the resulting hierarchy has depth at most two).

Under this set of assumptions, the only possible difference between the performance of two different hierarchies is at the one intermediate level in that hierarchy, since the performance of different hierarchies is necessarily the same at the root node and at the leaf nodes. Given the fact that the costs of all tests are equal, this difference is quantified solely by the value of each test; hence performing the test with higher value first is guaranteed to provide an optimal hierarchy. Which test is performed second is irrelevant as long as it successfully completes the diagnosis process.

Unlike the simplifying assumptions given in the first section of this chapter, this set requires that the hierarchy produced be very small. However, in one important sense this set of assumptions is a much more interesting set than the one given in the first section. It is possible for all these assumptions to hold and yet for one still to have an interesting application of the problem--one where the information-theoretic decision tree would be less desirable than the action-based hierarchy. The parachuting example given at the beginning of this thesis is an example of a problem that meets these assumptions.

In fact, the conclusion one can draw from this section is that the primary way in which action-based hierarchies might fail (and hence require experimental validation) is if the greedy approach in selecting the first few tests results in lower value actions further down the hierarchy, but still above the level of the leaf nodes. For that reason, when the experiments are done in the next chapter, and the number of faults is a power of 2, the minimum number of faults in a hierarchy will be 8. This is because the results of this section already formally validate the approach for a 4-fault hierarchy.

Section 5.3 -- Test Promotion

The final observation that can be made is that running the test promotion algorithm always either improves hierarchy performance or does not make it any worse. This observation is quite easy to see. Whenever a test promotion is potentially going to be made, the algorithm always looks to see whether it will improve or degrade hierarchy performance in the section of the hierarchy it affects. Promotion is only done if performance is going to be improved. Because each step of the algorithm either improves hierarchy performance or does not change it, running the algorithm is guaranteed to not worsen performance.

This observation should not be taken to mean that test promotion finds the best possible way to time the performance of tests. There are certain configurations that it never looks at. However it is guaranteed, by doing a full analysis of the effects of promoting tests, to not worsen things. Furthermore, from the observations in the last chapter, the complexity of test promotion is generally smaller than that of the basic hierarchy structuring algorithm, so it becomes a reasonable add-on to the algorithm.

Chapter 6
Abstract Experiments

As we saw in Chapter 5, the set of conditions under which we can formally prove that action-based hierarchies provide the best performance is quite limited. Thus, in this chapter and the next, we shall seek other means of validating this work. In fact, there will be three separate approaches described in these two chapters. In this chapter, the values of the inputs to the hierarchy structuring algorithm will be assigned randomly without regard to any specific domain, and the performance derived by using action-based hierarchies versus decision trees will be compared. If the performance of action-based hierarchies is found to be better by a statistically significant margin than that of decision trees, then this will be evidence in favor of using this approach for structuring hierarchies in real-time domains.

The validation method described in this chapter can both be criticized and praised on the grounds that it is independent of any particular domain. To answer that criticism, in the next chapter we will explore validation of the approach in a particular domain--intensive care unit patient monitoring.

Specifically, in this chapter we wish to evaluate the following two hypotheses:

❶ Using the action-based hierarchy will provide substantially better performance than the decision tree when evaluated as described in chapter 2.

❷ Using the decision tree will provide substantially better performance when only speed in reaching a leaf node matters.

Section 6.1 -- Experimental Design

The problem size, prior probabilities, test outcomes, and values of actions for faults were assigned as follows:

Problem Size-- The problem size was allowed to vary and was equal to the number of faults, actions, or tests in the domain. That is, the number of faults, actions, and tests were all kept equal, and this number is what was allowed to vary.

Prior Probabilities-- The difficulty in assigning prior probabilities is that although they must be assigned randomly, they must also sum to 1. The following method was used to assign a set of n prior probabilities (for n faults) $p_1, p_2, ..., p_n$: First a set of n-1 random variables uniformly distributed on the interval [0,1] was assigned by a random number generator:

$$x_1, x_2, ..., x_{n-1}$$

Then these numbers were permuted so that they were in non-decreasing order:

$$x_{i_1} \leq x_{i_2} \leq ... \leq x_{i_{n-1}} \text{ where } \{i_1, i_2, ..., i_{n-1}\} = \{1, 2, ..., n-1\}$$

and finally the priors are assigned as follows:

$$p_1 = x_{i_1}, p_2 = x_{i_2} - x_{i_1}, ..., p_{n-1} = x_{i_{n-1}} - x_{i_{n-2}}, p_n = 1 - x_{i_{n-1}}$$

The important property of this distribution is that the distribution for each prior probability generated in this way is the same.

Tests-- The number of tests, n, must be even. For each test, the fault set is randomly partitioned into two disjoint subsets of size $\frac{n}{2}$. The partition is assigned randomly from among the $\binom{n}{n/2}$ possible ways of doing this.

Values of Actions-- The value of an action for a fault is uniformly distributed in the interval [0,1].

In performing these experiments, we added another hierarchy structuring heuristic in addition to the basic information-theoretic and action-based heuristics: a random heuristic. The random heuristic selects a test at random from the set of all *relevant* tests that can be performed at a given node. Formally, this heuristic is defined as follows:

$$R(t) = \text{Uniform}[0,1] \text{ if } \exists o \in t\text{: } \emptyset \subset o \cap F_n \subset F_n$$

$$R(t) = 0 \text{ otherwise}$$

Section 6.2 -- Statistical Validation

In order to make any claims about the statistical validity of the results derived from these experiments, we need some means for computing statistical validity. The method used will be something called the t-statistic [HO47]. The idea behind the t-statistic is that we observe that a particular variable, say t_1, has values greater than those observed for t_2. We wish to be able to claim that the observed difference in value between t_1 and t_2 is statistically significant. The t-statistic provides a means of doing so provided that we know the difference $t_1 - t_2$ is bounded. If the number of observations of t_1 and t_2 is n then the t-statistic is given as follows:

$$t = \frac{\overline{t_1} - \overline{t_2}}{\sqrt{2}\sigma/\sqrt{n}}$$

where σ^2 is the variance of the difference $t_1 - t_2$. What this tells us is the number of standard deviations above the mean the observed difference between t_1 and t_2 is. For example, if the observed t-statistic is 2, then we can say with 97% confidence that the observed difference between t_1 and t_2 is statistically significant.

This concept can easily be applied to the current problem. If the variables V_1 and V_2 represent two different sets of values, one corresponding to each of two hierarchy structuring heuristics--the method for computing the value performance of the agent was described in chapter 2--then the standard deviation of the difference $V_1 - V_2$ is indeed bounded (by 0.5) so we can apply the t-statistic and compute the statistical significance of the observation that V_1 values are typically larger than V_2 on average. For example, for $n = 100$, the t-statistic will be approximately 14.1 times the mean difference between V_1 and V_2, meaning that a difference of 7% corresponds to one standard deviation above the mean and 14% corresponds to two standard deviations above the mean.

Actually, it could be argued that one can do much better than this. Because with very high probability all values in the abstract experiments will be above 0.5 and based on uniform distributions, it is reasonable to suppose that the standard deviation will not exceed 0.125, which corresponds to a difference of 1.75% being one standard deviation above the mean and 3.5% being two standard deviations above the mean.

We are also in the advantageous position of being able to simply run further trials if the existing observations do not provide statistically significant results.

The t-statistic can also be applied to evaluate the claim that a particular hierarchy structuring heuristic is better for a particular value of the deadline, although not necessarily for all values of the deadline. Looking at the $V(t,f)$ function defined in chapter 2, we can use it to give hierarchy performance at a particular deadline value t:

$$V(t) = \sum_{f \in F} V(t,f)P(f)$$

We can also use the t-statistic to evaluate the claim that a particular hierarchy structuring algorithm provides performance that is at least a certain percentage better than another heuristic. For example, suppose we wish to claim that the set of values V_1 is at least 20% better, on average, than the set of values V_2. Then we can compute a new variable $V_3 = 1.2\ V_2$ and then compare the values of V_1 and V_3 using the t-statistic.

Section 6.3 -- Experiment Results

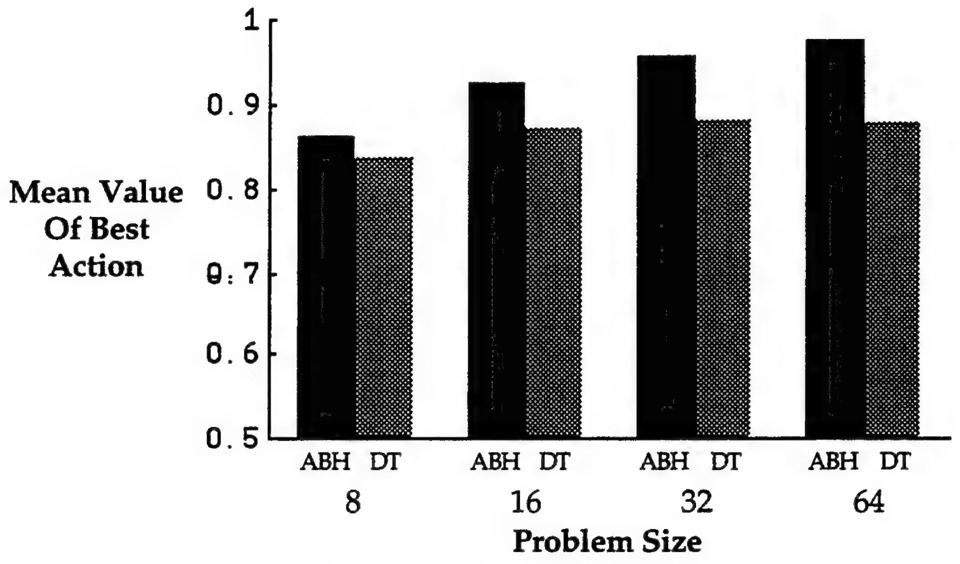


Figure 1: Hierarchy Performance Based on Problem Size

Figure 1 shows the results of the experiments for problem sizes of 8, 16, 32, and 64. Each hierarchy structuring heuristic was run on the same 100 randomly chosen problems, and the results averaged. As can be seen, the action-based hierarchy (ABH) consistently outperforms the decision tree (DT). The results were produced by taking the average performance of hierarchies up to the maximum time taken by either method to diagnose a fault node; deadline distributions were not taken into account in this set of experiments. It should be noted that a baseline of 0.5 rather than 0.0 is used. Since the values of actions are randomly chosen in the interval [0,1], it is rare for an action to have average value much less than 0.5 over a group of faults.

In this set of experiments, the costs of tests were uniformly assigned to be in the interval [0,1]. A couple of observations can be made: first, the relative advantage of using action-based hierarchies seems to increase as the problem size gets larger. Second, although action-based hierarchies give a consistently better performance than decision trees, the advantage is not dramatic. The first observation can perhaps be explained by the fact that there are only so many ways to structure a small hierarchy, and there is a high likelihood that purely by chance the decision tree will look much like the action-based hierarchy, and hence there will be no difference in performance. The second observation is likely caused by the fact that we are averaging over all possible values of the deadline. Action-based hierarchies will offer the greatest advantage for intermediate values of the deadline--something we will examine in more detail below.

The next set of experiments looks at how decision trees and action-based hierarchies fare when compared to hierarchies that are simply constructed randomly (all the tests that help in diagnosis are equally likely to be selected at a given node). In this set of experiments, the costs of tests were all assumed to be constant (equal to 1), which would

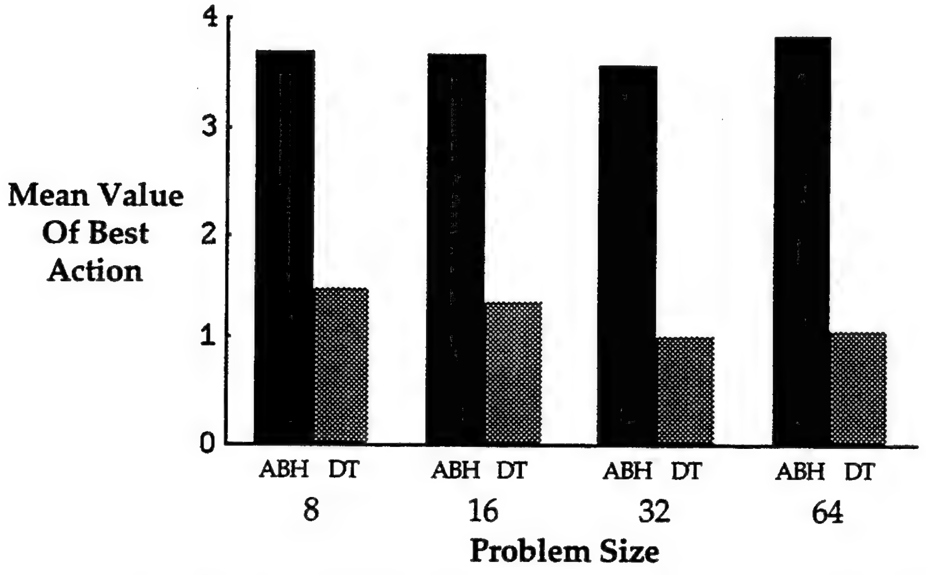


Figure 2: Hierarchy Improvement vs. Random Structuring

tend to reduce the advantage of using action-based hierarchies. We compared the percentage improvement of using action-based hierarchies relative to random hierarchies versus the percentage improvement of using decision trees relative to random hierarchies. The results are shown in Figure 2. Note that although the percentage improvements tend to be small, action-based hierarchies provide a vastly better improvement rate than do decision trees. Increases in problem size appear to have little effect on the value of action-based hierarchies, while decision trees seem to get somewhat worse as the problem size increases.

Figure 3 shows the difference in performance, over time, of action-based hierarchies, decision trees, and the randomly structured hierarchies. This shows that action-based hierarchies always do at least as well as the other approaches, no matter what the value of the deadline (the amount of resources available for diagnosis), but that this advantage varies over time from nil at time zero, to a maximum for deadline values of around 3 or 4, back to nil for large deadline values. This corresponds, roughly speaking, to the fact that all hierarchies perform equally well at their root and their leaf nodes, but that at the intermediate nodes there is an advantage for certain hierarchies over others.

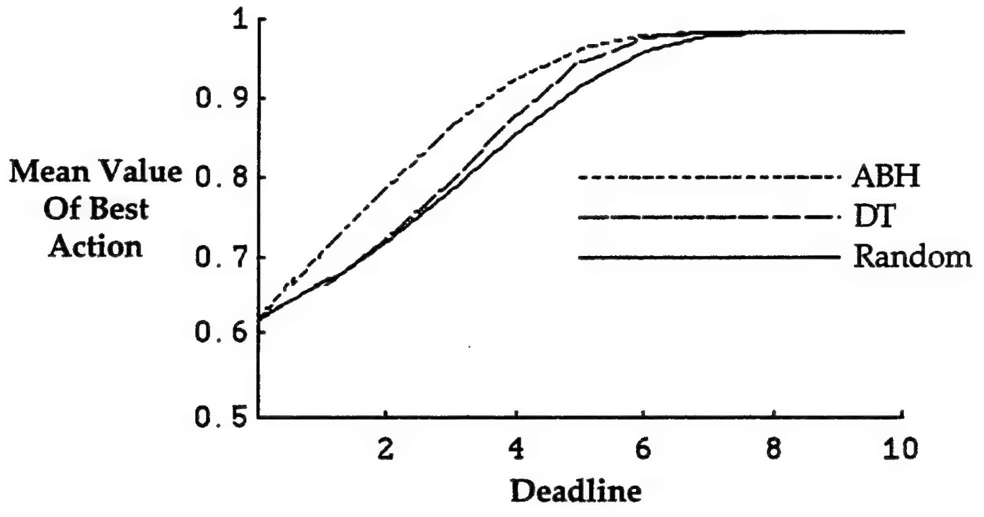


Figure 3: Hierarchy Performance with Different Deadlines and Heuristics

The reader may wonder why there are not certain time points at which decision trees would have an advantage over action-based hierarchies. The answer appears to be that in the profile of the performance of a single triple of hierarchies (ABH, DT, Random) over a single fault, there likely *would* be an advantage for decision trees at certain time points--namely those time points where the decision tree has completed its diagnosis but the action-based hierarchy has not done so. However, the numerical value of those time points varies from one instance to another, and when we *average out* as shown in the above diagram, there is no fixed value of the deadline where decision trees have a consistent advantage. In the next chapter, there will be examples of cases where the performance of decision trees is noticeably--although not greatly--better than that of action-based hierarchies.

So far we have looked only at the advantage offered by action-based hierarchies when we evaluate the performance based on partial results at intermediate nodes. Suppose that the only thing that counts is getting to a leaf node as quickly as possible. Here, we can define another evaluation criterion different from the one defined in chapter 2. This evaluation criterion, known as the *leaf-node evaluation criterion*, computes the probability that a leaf node will be reached by a certain value of the deadline. This criterion may be computed as the weighted average of its value for each fault:

$$V_l(t) = \sum_{f \in F} V_l(f,t)\ P(f)$$

The value for a given fault may be defined by starting at the root node and time zero:

$$V_l(f,t) = V_l(f,t,R,0)$$

and the value for a node $n = <C,t,a,F_n>$ may be computed recursively: in the case where $F_n = \{f\}$,

$$V_l(f,t,n,E) = 1 \text{ for } t \geq E$$

and in all other cases assume there are k outcomes of the test t, $\{o_1, o_2, ..., o_k\}$, such that $f \in o_i$. Then if the corresponding children nodes are n_i for $1 \leq i \leq k$, we get

$$V_l(f,t,n,E) = 0 \text{ for } E \leq t < E+C(t)$$

$$V_l(f,t,n,E) = \frac{1}{k}\sum_{i=1}^{k} V_l(f,t,n_i,E+C(t)) \text{ for } t \geq E + C(t)$$

In this case, we would expect decision trees to outperform action-based hierarchies because this is the problem that decision trees are designed to solve. The question is, by how much will they outperform action-based hierarchies? In Figure 4, we show a graph of the probability of success versus deadline. By this we mean the probability that a leaf node has been diagnosed by the deadline. Looking at this figure it is apparent that over a relatively small range of the deadline (between about 4 and 7) using a decision tree offers a very significant advantage over the other approaches. However, for other values of the deadline, it makes very little difference. We can conclude from this that if diagnosing down to a leaf node is especially important and if we expect an intermediate value for the deadline, it would be better to use decision tree structuring. In all other cases, it would seem better to use action-based hierarchies.

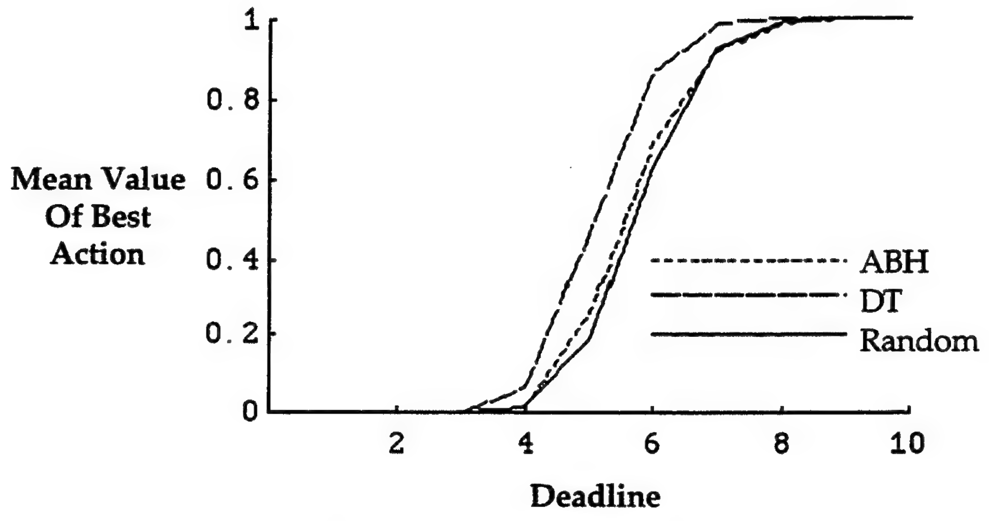


Figure 4: Hierarchy Performance using All-or-Nothing Evaluation

Section 6.4 -- Experiments with Deadline Distributions

The previous set of experiments ignored deadline distributions. We next run a set of experiments with deadline distributions taken into account. The hypothesis we wish to test is the following:

- Using the deadline distribution heuristic described in Chapter 3 will provide a better performing hierarchy than either of the regular action-based or information-theoretic heuristics, as well as better than the random heuristic.

The experiment specifications for this set of experiments is the same as for the previous set, except that now we need to specify deadline distributions instead of ignoring them. First, we define the *expected complete diagnosis time* as the average cost of a test multiplied by the average depth of the hierarchy; that is if the number of faults is n, then this number is

$$C(t) \log_2 n$$

Now each fault was randomly assigned to have one of nine possible deadline distributions: uniform distribution with the minimum 0 and the maximum either 0.5, 1.0, or 2.0 times the expected complete diagnosis time; exponential decay with a half life either 0.5, 1.0, or 2.0 times the expected complete diagnosis time; or the deadline is known in advance to be exactly 0.5, 1.0, or 2.0 times the expected complete diagnosis time.

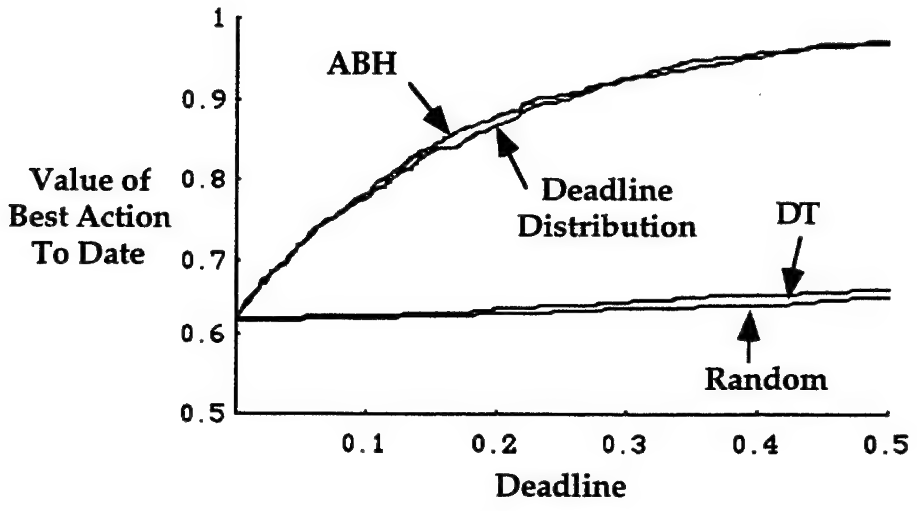


Figure 5: Hierarchy Performance with Non-trivial Deadline Distributions

The results are shown in Figure 5. The results may be a little surprising. First, as before action-based hierarchies are substantially better than decision trees or random structuring. However, there was no appreciable difference between structuring taking deadline distributions into account versus not taking them into account. A clue to the answer lies in the fact that the difference between action-based hierarchies and decision trees was much greater than in previous examples. The reason for this was that there are 64 tests, and costs of tests were allowed to range uniformly between 0 and 1 in this set of experiments. Hence, there were enough low cost tests to allow diagnosis to be completed very quickly when costs are taken into account, but conversely when costs are *not* taken into account, diagnosis is much slower. The deadline distributions were computed by taking an average test time of 0.5

and 6 tests to do complete diagnosis, for an average diagnosis time of 3.0; hence significant changes in deadline distribution do not occur until half this or 1.5 time units out.

In other words the deadline distributions turned out to be too long for this set of experiments for us to see any significant effect. One may wonder what the point is in giving this result at all. It *is* an interesting result in the sense that it shows that even in cases where deadline distributions do not have a significant bearing on the final result, taking them into account does not worsen hierarchy performance in any way. Thus, it would be an interesting topic for further research to determine whether deadline distributions significantly affect hierarchy structuring under any circumstances and if so under what circumstances, taking into account the analysis of the previous paragraph. All experiments that were run, however, showed similar results to those shown in Figure 5.

It does not follow that deadline distributions are an unimportant part of the analysis. Indeed, deadline distributions are essential to the process of *test promotion*, and the next section will show that test promotion can lead to a significant improvement in hierarchy performance.

Section 6.5 -- Experiments with Test Promotion

In the next set of experiments, we look at the relative performance of hierarchies when physical costs of tests are taken into account and tests are potentially promoted to higher level nodes in the hierarchy. Specifically, we ran the same set of experiments as previously, but with each test assigned a *physical cost* of 0.02 as well. Also, the definition of performance of the hierarchy changes somewhat when we take into account physical costs of tests, because now it is necessary to subtract the cost of all tests performed to date from the value of the hierarchy, as described in chapter 2. As before, the deadline distribution heuristic was used before doing test promotion, on the basis of the fact that it is impossible to do test promotion without deadline distribution. The results of this set of experiments is shown in Figure 6.

The results show that for intermediate values of the deadline, it is better to use test promotion. For short values of the deadline, the large number of initial tests requested by the test promotion algorithm results in a net loss in values, since many of those tests will not have come back in time to be useful.

For long values of the deadline, it would have been more efficient to perform fewer tests, because there was sufficient time for them to come back, so the unnecessary tests result in a greater cost. It is for the intermediate values of the deadline that one can observe a significantly better performance by using test promotion.

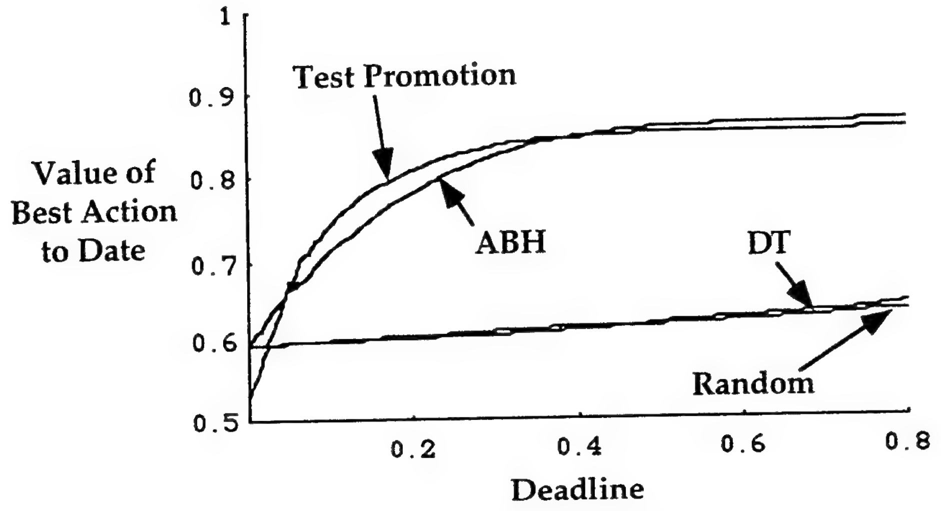


Figure 6: Hierarchy Performance with Test Promotion

This should not be construed to mean that the user of the reactive planner has to estimate whether the deadlines are likely to be intermediate in value, and only do test promotion in this case. As long as the deadline distribution values are reasonable, this is handled automatically. The reactive planner will converge on a hierarchy that behaves similarly to the one given above only if it is more beneficial to perform well for intermediate values of the deadline than it is costly to perform not as well for the large and small values of the deadline.

Chapter 7
Domain Experiments

The experiments described in chapter 6 were successful as far as they went. They provided a good preliminary validation of the approach of using action-based hierarchies as opposed to decision trees or randomly structured hierarchies, as well as the increased benefits that can be realized by taking deadline distributions and test promotion into account. As such, they significantly extended the scope of the theoretical analysis, which was extremely limited in that it required very strict assumptions in order to be provably optimal. However, to date the approach has been completely in the abstract--in this chapter we apply the approach to a real-world problem to show that similar results can be realized in such a problem.

The first part of this chapter will describe the domain of application--surgical intensive care unit patient monitoring--and then results from applying the approach to this problem will be presented. These results, while encouraging, will not be subject to the same statistical validation that the results in chapter 6 were subject to because there will be essentially only one hierarchy generated. In the second part of this chapter, we will analyze the problem to determine how it differs from the abstract problems that were solved in chapter 6, and then we will run further experiments on abstract problems with the same general structure as the medical problem. The purpose of this additional set of experiments is to ensure that there is neither anything in particular about the domain, nor about the distributions of variables used to run the experiments of chapter 6, that resulted in the encouraging results.

Section 7.1 -- Intensive Care Unit Patient Monitoring

The domain to which these ideas are applied is that of monitoring a patient in an intensive care unit (ICU). A patient in the ICU typically has multiple organ failure and is placed on a ventilator to assist with breathing. Because of this, there can be many problems that might arise that require fast response; this domain is attractive because of the need to provide such response, and because there are often several different responses available of differing specificity that can be undertaken given a differential diagnosis of different sizes.

The areas requiring immediate response can be broadly categorized into three areas: anemic hypoxia problems which generally require a transfusion of some type; oligemic hypoxia problems which may require the immediate invocation of an ACLS (advanced cardiac life support) algorithm or treatment of the underlying cause if more time is available; and oligemic hypoxia problems generally requiring either some tweaking of the ventilator settings or treatment of the underlying cause of the problem. Corrective actions in this domain can be of several types: they can buy a medical practitioner time in order to diagnose the real underlying problem; they can be effective but not necessarily definitive for a wide range of problems; or they can represent the definitive therapy for a particular fault. Because the structuring algorithm requires a particular number for the value of each action, a medical domain expert provided a heuristic estimate of the numerical value of each action for each fault on a scale of 0.0 to 1.0. In general, definitive therapies were assigned a value of 1.0; therapies valid for a wide range of problems were assigned somewhat lower values ranging from perhaps 0.5 to 0.85; therapies that merely buy time were assigned values that were lower still--usually under 0.5.

The tests available in this domain tended to fit into two large categories--lab tests that require an average of 20-30 minutes to come back--and monitoring actions involving checking parameters--these could be completed in under a minute. There is also a third type of test that requires the insertion of a monitoring device such as a Swan catheter. To make an initial measurement of the value of such a parameter is therefore quite expensive, but additional measurements then become quite inexpensive. The therapist must therefore take into account both the initial cost and the possible future benefit in deciding whether to perform such a test.

The actual tests in this domain require a minimum of computational complexity to be analyzed; this is not a limitation on the action-based hierarchy approach but rather is an artifact of this particular domain.

Many of the inputs for structuring the algorithm will depend on the type of patient in the ICU. For the purposes of illustration, the values used in this demonstration were given by a domain expert based on a typical patient at the Palo Alto Veteran's Administration hospital--a 67-year-old male who had just undergone coronary artery bypass graft (CABG) surgery.

Various medical domains have been typical vehicles for applying AI ideas over the years. For example, the KARDIO system [BR87] precompiles ECG descriptions based on model-based reasoning, which it uses to classify arrhythmias at runtime. This represents a solution to half the problem that is solved in this thesis; we precompile solutions but also can make tradeoffs at run-time in cases where a complete diagnosis is impossible. This appears to be impossible in KARDIO. Another system, VM [FA80], is similar to the present work in that it is a real-time medical AI system, but it handled real time in a different manner than here. In particular, it was concerned with the complexity of reasoning algorithms and with when data values became stale over time, but not with deadlines. Therefore, it was unable to make the necessary tradeoffs that a system using action-based hierarchies can. Another medical system, TraumAID [CL89] was able to handle incomplete data, but the notion of meeting a deadline is missing.

Specific to the problem of intensive care unit patient monitoring, there is a fairly wide body of literature. Factor and colleagues [FA90] developed an architecture known as the *process trellis* and applied it to the problem of ICU patient monitoring. The process trellis consists of a graph of decision processes which execute in parallel; it has the ability to provide parallelism even when knowledge acquisition is done by domain experts with little knowledge of parallelism. Although the ability to do parallel processing would seem to be a major advantage to this approach, the fact that the processes modeled all directly correspond physiologically to the domain restricts its usefulness in responding to deadlines. Response on deadline requires a component that knows specifically about deadlines and not necessarily about specific organ systems.

An ICU patient monitoring system with a strongly decision theoretic component is VentPlan ([RL93], [RL91]). The idea here is to select a decision-theoretic model of physician preferences that effectively trades off complexity and accuracy. This is a different tradeoff than the one made by action-based hierarchies, which trades off time for specificity. Still, it would be interesting to compare the two approaches to see which yields the better results. It appears plausible that action-based hierarchies would perform better in cases where incomplete information is available, while conversely

VentPlan would perform better when complete information is available but time is short.

EINTHOVEN [WI91] is another ICU monitoring system devoted chiefly to the interpretation of ECGs. As such it attacks a different subproblem of the ICU problem than the one discussed in this thesis. Other ICU monitoring systems include WEANPRO [TO91], SIMON [UC93], and the work of King and colleagues at Vanderbilt [XU92].

Section 7.2 -- Analysis of the Domain

In order to make the experiments of chapter 6 more realistically reflect the medical domain, we first analyzed the medical problem on a number of dimensions to compare its properties to the abstract distributions we used in the previous experiments.

7.2.1 Prior Probabilities of the Faults

In the chapter 6 experiments the prior probabilities for a set of *n* faults were selected by selecting a uniform distribution for the *n-1* partitions between the faults. This results in the standard deviation for the prior probabilities of the faults being

$$\frac{1}{n}\sqrt{\frac{n-1}{n+1}}$$

This means that the expected standard deviation for the *58* faults in the medical KB is *0.01695*. For comparison purposes, the actually observed standard deviation of the fault priors is *0.01536*. So in the case of prior probabilities, there does not appear to be much difference between the distribution observed in the medical problem and the distribution used in the experiments. Therefore, in redesigning the experiments, we made no changes in the distribution for fault probabilities.

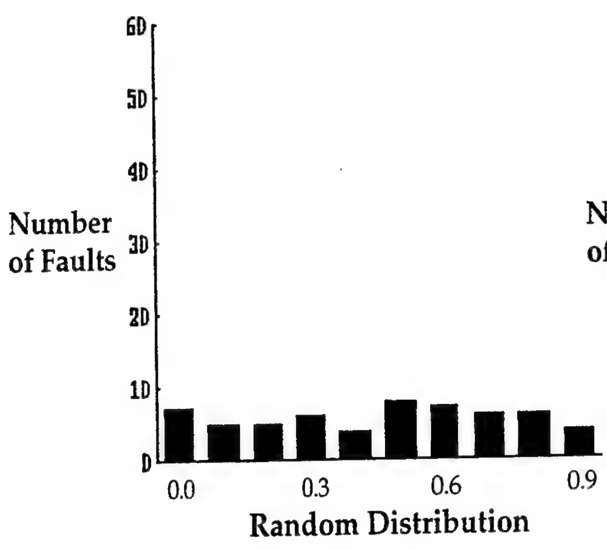
7.2.2 Values of Actions for Faults

In the experiments, the value of the actions for each fault is uniformly distributed on the interval [0,1]. In order to analyze the distribution of the values for a given action, we divide this interval up into k subintervals $[\frac{i}{k}, \frac{i+1}{k})$ and assign the number 1 to the last subinterval. We look at how many of the faults fall into each of these subintervals. We would expect, in the theoretical experiments, that the distribution of the number of faults in each interval would be Poisson, with mean $\frac{n}{k}$ where n is the number of faults. Hence we would expect a standard deviation of $\sqrt{\frac{n}{k}}$. In the actual domain, we have $n = 58$, and in analyzing we use $k = 10$, so we would expect the standard deviation to be *2.408*. In the actual domain, the standard deviation averages *17.142* whereas the maximum possible standard deviation would be *17.4*. What we are observing here is illustrated in Figure 7.1. The left hand side of the figure shows, for a typical action, the type of distribution of the values of the action for the 58 faults that would be observed if the values were assigned randomly. The right hand side shows what actually happens in the domain. In the domain, there tend to be many faults for which a given action has value 0, and a much smaller number for which it has value 1. For some actions, of course, there will be intermediate values as well, but we see here is fairly typical. In other words, the standard deviation is about as high as it can possibly be, which is what we observed in computing its value.

We will observe a similar result if we compute the mean, over all faults and actions, of the values. We would expect in the experimental domain that this mean would be 0.5. In actuality, the mean is *0.01108*. Thus, the average value of actions is far below that which occurs in the experiments.

7.2.3 Costs of Tests

In the experiments, the costs of tests is uniformly distributed. In the actual medical domain, it appears that the costs of tests is less uniformly distributed. There are only a few different values (e.g. 1 min, 5 min, 20 min, 30 min) that it ever attains, and these values are clearly not uniformly distributed in the interval from 0 - 30 min.



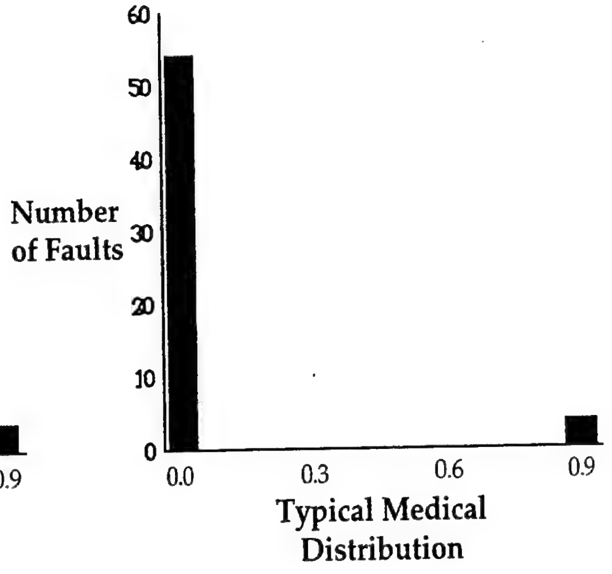


Figure 7.1: Distribution of Action-Fault Values: Random vs. Domain

7.2.4 How Tests Divide up the Fault Set

In the experiments, each test divides the fault set exactly in half, and the two sets are disjoint (all tests are binary). In the actual domain, we observe that the tests split up the domain in the following way:

Outcomes	Num. Tests	Average Size of Outcomes
2	27	54.59 47.89
3	3	57.00 57.00 56.00
4	10	56.00 53.00 52.50 51.50
5	2	54.50 53.00 51.00 50.50 49.50

6	1	56.00 54.00 53.00 53.00 53.00 52.00
8	1	51.00 50.00 46.00 45.00 44.00 43.00 39.00 38.00
17	1	50.00 49.00 41.00 41.00 41.00 41.00 41.00 41.00 40.00 40.00 40.00 40.00 40.00 40.00 40.00 39.00 38.00

What the numbers in this table signify are best shown by example. In the first line of the table, we see that there are 27 tests with 2 outcomes (the first set of experiments assumed *all* tests were binary). For each of those 27 tests, there will therefore be 2 associated subsets of the set of all faults corresponding to each of the 2 outcomes. The larger such subset has, on average, 54.59 faults, and the smaller has on average 47.89 faults (out of a total fault set of 58 faults).

One thing we can conclude from this table is that the tests tend to have more outcomes than in the domain experiments, and they tend to be less discriminating (they reveal less information about the correct fault). A realistic set of experiments would therefore have to incorporate this feature.

Section 7.3 -- Experiments in the Domain

The hierarchy structuring algorithm was run, with both action-based and information-theoretic heuristics, on the medical problem outlined in section 7.1. Although referred to as experiments to match the terminology used elsewhere in this thesis, this aspect of the work really amounts to a complete analysis of the approach as it applies to this problem. The reason is that it is quite easy, computationally, to measure the performance of the hierarchies generated on each of the 58 faults individually, and then weight the performance on the basis of the prior probabilities of the faults. Of course, the fact that this complete analysis is possible does not mean that there is no uncertainty in the domain, so in that sense we are solving a simplified version of the actual problem with the action-based hierarchies or information-theoretic hierarchies. However, given this simplified approximation to the real problem, a complete analysis of the value of a particular hierarchy is possible, and that is what we have done with the experiments.

The basic assumption that is made in moving from the actual problem to the simplified problem is that the various inputs to the hierarchy structuring algorithm are fixed and known in advance. For example, a given test is assumed to come back from the lab in a fixed amount of time; in the actual domain the time would be uncertain and known only to have a certain approximate mean. Similar comments may be made about the other inputs to the hierarchy structuring algorithm.

It should also be remembered that the goal of this research is to compare different possible approaches to the same general problem of reacting when faced with limited resources and only basic associative knowledge about the domain. Other approaches such as model-based reasoning (handling first-principles domain knowledge better) or belief networks (handling uncertainty better) might perform better in certain cases especially when the agent has greater resources with which to complete its diagnosis and has the necessary domain knowledge. It is not the goal of the present research to make this comparison, although it would definitely be an interesting topic for further research.

Figure 7.2 shows a small part of the action-based hierarchy generated for this particular domain. In this figure, the leaf nodes are shown with the specific faults attached, and the higher level nodes have associated fault sets that are the unions of the fault sets for all lower nodes. The associated actions are indicated for all nodes. For example, at the top node of this subhierarchy, the action of transfusing red blood cells conditionally was found by the structuring algorithm to be a good action for all the lower nodes, and so is the action of choice at that level. At a lower level, when the specific problem has been diagnosed, it may be necessary to consider surgery, but only when more information becomes available.

It should be remembered that the agent that uses this hierarchy is not a planning system. For example, what is indicated above as the simple action of "consider surgery" would translate to a plan that included performing a transfusion and then performing surgery. The full details of actions are not known to the hierarchy structuring algorithm because they are not of interest to it. The action is given in the simple form of merely considering surgery because at the level of reacting, the agent is not interested in the details of an action but only in its value for given faults. What is of interest in the above figure for the current research is that if sufficient time is available, the agent will find the action of considering surgery, which, when executed in its entirety, will be superior to merely performing a transfusion. But if insufficient time is available, then the intermediate action of performing the transfusion will be much better than doing nothing.

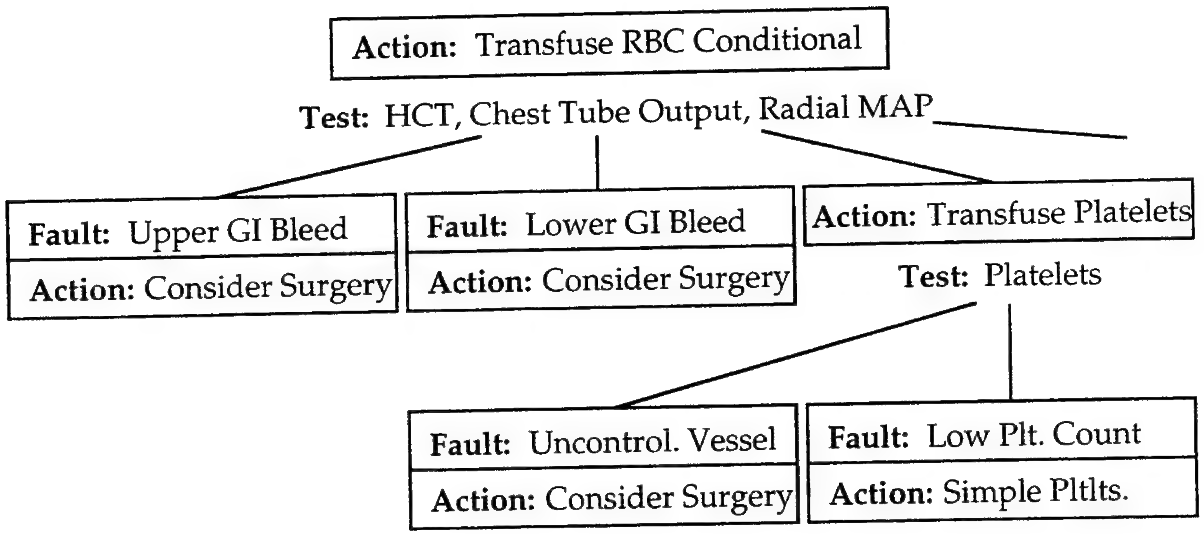


Figure 7.2: Portion of Medical Action-Based Hierarchy

With regard to the experiments themselves, there were two experiments run. Both compared the average value of the best action obtained, over all possible faults, as a function of time--the comparison was between the two different structuring heuristics--action-based and information-theoretic. In one experiment, the temporal cost of performing tests was not taken into account, and in the other experiment, it was. The reason for performing both experiments is that the classical decision trees do not take cost into account, and it would be an obvious improvement to that approach to take cost into account. Therefore, it is interesting to see how much of the improvement that might be gained by using the action-based heuristic is due to the fact that cost is taken into account, and how much is due to other factors.

Figure 7.3 shows the results of the experiment where test costs are not considered. The x-axis is just the number of tests performed, since no information about their costs is known to the hierarchy structuring algorithm. As can be seen, there is a modest advantage of action-based hierarchies for one and two tests, and with three tests there is a crossover effect where decision trees are better. Whether this is due to a mere artifact of the domain, or more because of the intrinsic advantage decision trees have over longer time intervals, is hard to say with only one data point.

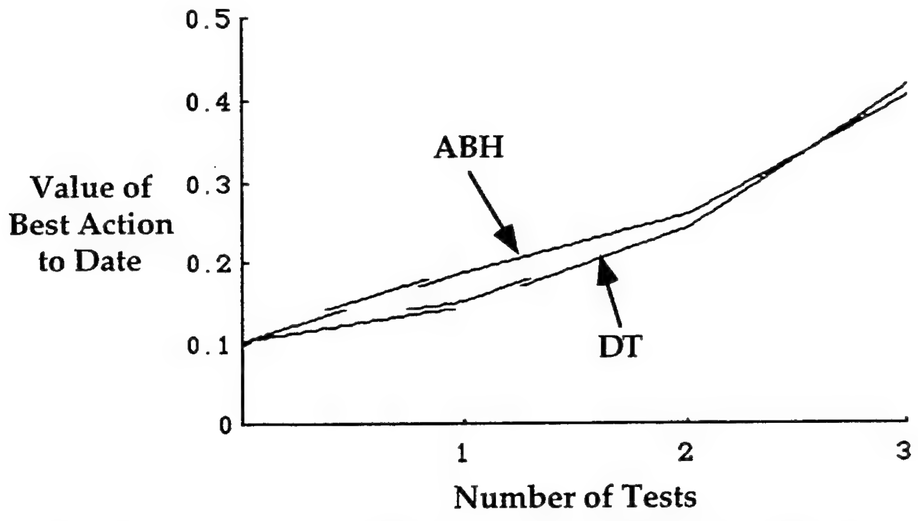


Figure 7.3: Hierarchy Performance on Medical Problem (no costs factored in)

Complete diagnosis takes longer than three tests, but for larger number of tests, the results are not too interesting--the two types of hierarchy structuring perform roughly equally. This fact tends to indicate that the advantage enjoyed by decision trees for 3 tests is a mere artifact of the domain.

The interpretation that one can draw from this experiment is that for a very small number of tests action-based hierarchies have an advantage over decision trees even when costs are not taken into account. For larger numbers of tests, they perform roughly equally, so this would give reason to prefer action-based hierarchies even in a domain where all test costs are equal, although the advantage would be small.

Figure 7.4 shows the same experiment run, but this time with the costs of tests factored in. As can be seen, this time action-based hierarchies perform much better than decision trees as the deadline increases. The conclusion that should be drawn from this combination of results is that while action-based hierarchies provide some advantage even without taking costs into account, clearly the majority of the improvement seen in figure 7.4 is due to factoring costs in.

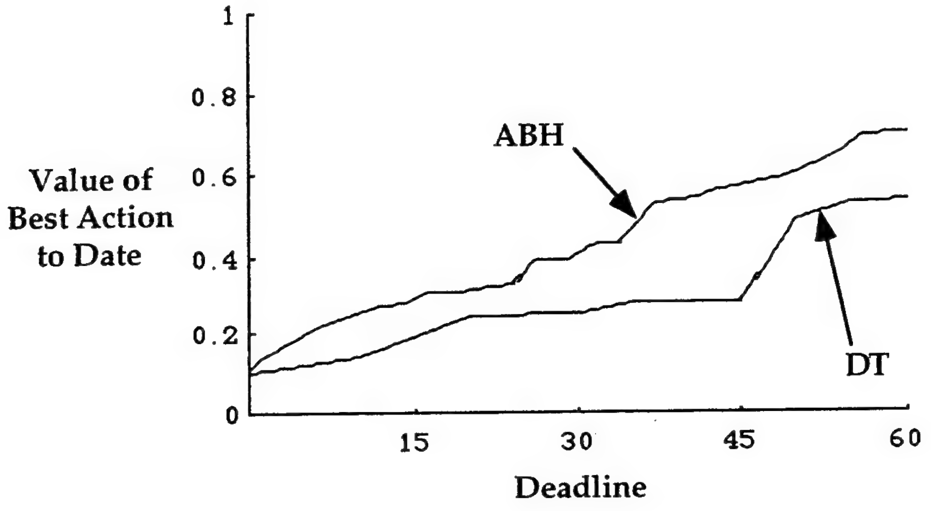


Figure 7.4: Hierarchy Performance on Medical Problem (costs factored in)

Section 7.4 -- Abstract Experiments with a Similar Structure

We have designed and run a set of experiments to test the hypothesis that action-based hierarchies provide a substantial improvement over decision trees when used in a problem domain with randomly assigned priors, values, costs, and tests, but that closely mimic the medical domain. Specifically, we assigned these numbers randomly using the following distributions:

Priors: A uniform distribution such that the sum of the priors is equal to 1 was used. This is the same distribution as used in the original set of experiments.

Values: The value of each action for each fault is independently assigned. With 90% probability, it is uniformly assigned to the interval [0,0.1], and with 10% probability, it is uniformly assigned to the interval [0.9,1.0].
Costs: Costs of each test were assigned independently. With 50% probability, tests were assigned to have the value 0.1667, and with 50% probability they were assigned to have the value 1.0000.
Tests: Each test was randomly assigned to have a certain number of outcomes, with each possible number of outcomes from 2 - 17 being assigned with the frequency that it occurs in the above table. Then, based upon the total number of faults, the number of faults in each outcome were assigned based upon the above table, subject to the condition that the maximum number of faults per outcome of a test is at least one less than the total number of faults. Finally, specific faults were assigned to each outcome of the test, subject to the condition that the union of the fault sets for all outcomes of a test must be the set of all faults.

As before, the experiments were run for problem sizes of 8, 16, 32, and 64 faults, test and action sets. However, it was computationally expensive to run the experiments and generate complete hierarchies, especially for the larger problem sizes. Therefore, we picked a fault at random and then randomly assigned outcomes to the tests that were consistent with this fault being present. We evaluated performance of the hierarchy solely on this fault for this set of outcomes; to do so did not require construction of the entire hierarchy. We repeated this process 100 times for different assignments of priors, values, costs, and tests. Because this approach is, in effect, 100 different simulations of the performance of 100 different generated hierarchies, it will be statistically valid.

7.4.1 Experimental Results

Figure 7.5 shows the results of this set of experiments. The results shown are what we would expect--action-based hierarchies perform better than decision trees, and decision trees perform better than random. The other expected result is that it takes a great many tests to get a complete diagnosis. Since there is so little information content in each test, this is probably to be expected. Most outcomes of most tests include the vast majority of faults. Still, this does not seem to capture the medical problem fully, so further investigation of this point is needed, as well as possible design of further experiments.

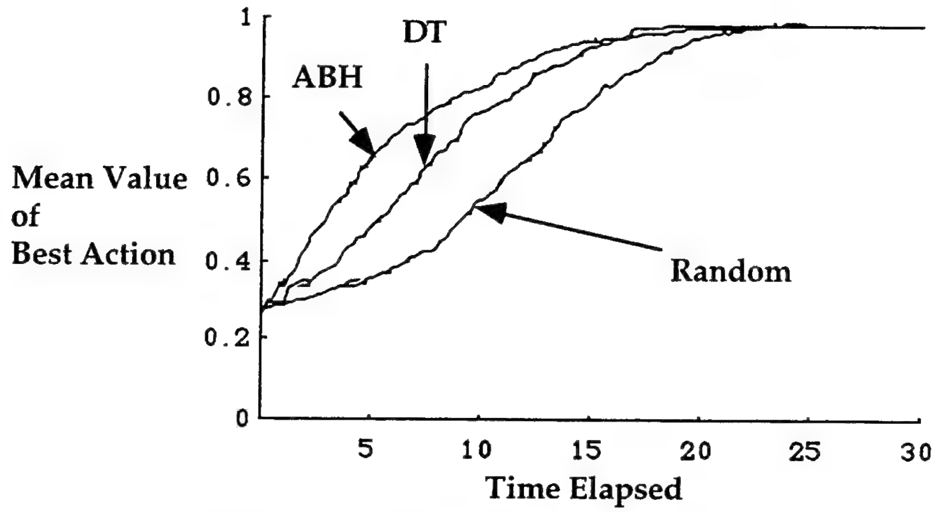


Figure 7.5: Hierarchy Performance (64 faults) Parameter values similar to medical domain

It also appears that a problem size of at least 64 faults is necessary before there is a significant difference between the different approaches to hierarchy structuring. The following figure shows the hierarchy performance for the 32-fault problem. As can be seen, similar effects to the 64 fault case are observed, but they are not significant with only 32 faults.

The biggest caution that must be added to the results of this set of experiments is that it is clear that only certain aspects of the domain have been captured in the analysis that led to the specifications of the experiments, and it is perhaps difficult to see how to modify the specifications to avoid this problem. In particular, the high computational complexity that was mentioned above with this set of abstract experiments occurs because none of the tests has much discriminating power among faults; hence a vast number of nodes is created. In the domain, this difficulty does not arise and the number of nodes remains manageable. The reason is that although on average tests in the domain have little discriminating power, there are a few tests that do have high discriminating power, and those tests tend to be performed at the high levels of the hierarchy.

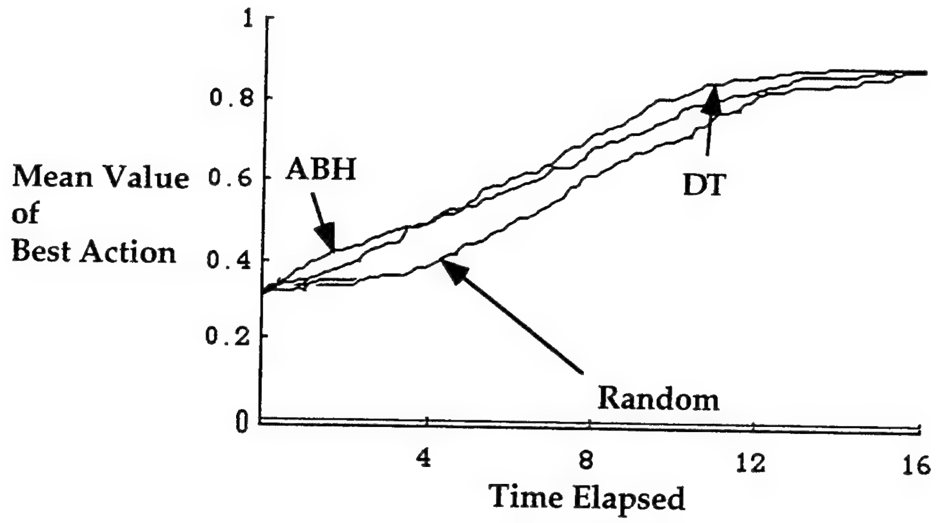


Figure 7.6: Hierarchy Performance (32 faults)
Parameter values similar to medical domain

However, this set of experiments does provide evidence that the approach is robust in the face of very different types of inputs. Coupled with the other experimental results, it provides a fairly strong endorsement of the approach.

Chapter 8
Implementation

In a certain sense, the rest of this thesis stands on its own without the need to discuss any implementations of the ideas. That is, one could use these ideas to build a reactive planner and an agent that would be able to respond in real time to faults even when complete diagnosis is impossible.

However, the ideas have been implemented in a running system, known as ReAct, and there are a couple of good reasons to include a discussion of this system in the thesis. The first reason is that a number of issues come up in the implementation of a system that do not come up in its design or in the analysis and experiments. A discussion of the implementation provides a good framework for leading into a discussion of these issues. Because the implementation is not the central point of this work, there is no claim being made here that the choices that were made in doing the implementation were optimal. However, by presenting the issues that arose and possible alternatives, the reader will better be able to make reasonable choices in applying the ideas.

The second reason for discussing the implementation is that it provides a framework for describing an example of a reactive plan in action. The nature of the analysis so far has made it impossible to see this, because the experiments presented overall average performances of the entire hierarchy.

The implemented system is known as ReAct. It consists of two parts--the reactive planner and the necessary knowledge sources to execute the reactive plan. The reactive planner is written in Lisp; the reactive plan is written in the BB1 blackboard architecture [HR85]. ReAct is part of the Guardian intensive-care unit patient monitoring system [HR92]. It is not, however, the whole of Guardian and is intended only to provide a response when the available more compute-intensive methods, such as the planner [WA90] and model-based reasoner [HE89] do not have sufficient computational resources to complete their tasks.

Like the other parts of Guardian and any system implemented in BB1, ReAct comprises a set of *knowledge sources* (KSes) each of which has expertise at performing a part of its task. When certain conditions are met, a KS is instantiated to produce a *knowledge source activation record* (KSAR). A *control plan* is used to determine which KSAR to execute next. The process of deciding which KSAR to execute next and executing it is referred to as a *BB1 cycle*.

This decoupling of the reactive planning phase (as a Lisp function) and the plan execution phase (in BB1) permits two different modes for the use of ReAct. First, the hierarchy structuring algorithm (forming the basis of the reactive planner) may be used and the resulting action-based hierarchy used in BB1 to provide real-time performance. However, it is also possible that a domain expert may feel that the basic framework is an acceptable one but that he/she can provide an action-based hierarchy by hand that will be more effective than the generated one. Hence, ReAct allows this possibility by simply allowing a knowledge base to be built up that is used directly in BB1.

Both the reactive planner and the plan execution phase are designed to be domain independent. That is, although the overall system (Guardian) is geared toward a specific domain (intensive care unit patient monitoring) both parts of ReAct could be used as part of a system in any domain. The inputs to the hierarchy structuring algorithm are the only thing that is domain dependent.

There is no direct mechanism allowing for dynamic replanning where new reactive plans could be constructed based on changing conditions in real time. However, ReAct allows more than one action-based hierarchy to be in existence at any one time. Hence, these multiple action-based hierarchies could be created by a BB1 knowledge source outside of ReAct. Furthermore, it is not necessary that all action-based hierarchies in existence be active at any one time, so other knowledge sources (outside ReAct) could also change the set of action-based hierarchies that are active at any one time. Thus, dynamic replanning can be accomplished by having another knowledge source run the hierarchy structuring Lisp routine and making active the resulting action-based hierarchy.

ReAct is intended to be closely coupled to another system designed by Dabija [DA94]. Dabija proposes that the set of faults (or contingencies) that the system can respond to depend on a number of factors such as the urgency (either too urgent or not urgent enough provides reason not to react), consequences and side effects of acting, and others. The intent therefore is that Dabija's algorithm be run first and the resulting set of faults be used as an input to the hierarchy structuring algorithm. This also provides a framework for doing dynamic replanning, since if the circumstances change such that the set of faults to which the system should be responding change, then it is necessary to generate another action-based hierarchy.

Because ReAct is running as part of a larger system and not as a separate entity unto itself, some issues that otherwise would not need to be confronted need to be addressed. For example, although the hierarchy structuring framework provides for tests to be performed at a particular time, it is entirely possible that another part of the system may perform one of these tests out of the sequence that ReAct is expecting it to be performed in. One way this may happen is that certain parameter values may be monitored on a regular basis, and if these parameters correspond to particular tests in the system, then this amounts to the performing of certain tests on a regular basis. In fact, a particular subsystem, Focus, is concerned with just this issue--how frequently should parameter values be monitored and sent to the agent [WA89]? A related issue is when a search of the hierarchy should be initiated. The entire analysis assumes that it simply happens at time 0--so the question is how one defines time 0. The definition given earlier in this thesis states that time 0 is when evidence is first noticed about a problem or fault. Although this is reasonable, it tends to assume that only a single instance of ReAct will be run. Clearly this is inappropriate in general so the issue becomes one of deciding when to instantiate ReAct to run on another problem.

Another issue that arises is a larger version of the dynamic replanning problem. If dynamic replanning is to be done, it means that the inputs to the hierarchy structuring algorithm may change from time to time. The problem being addressed by this thesis does not concern itself with how these inputs change. As was mentioned above, Dabija concerns himself with how a particular input to the algorithm--the set of faults to be responded to--changes in real time. Someone wishing to use this system and do dynamic replanning needs to consider the question of how all inputs to the algorithm might change over time, and if so, how they are likely to change.

Appendix B gives a complete programming manual for the implementation; what follows in this chapter is not intended to be a programming manual but rather a description of how the more interesting aspects of the implementation affect the developer of a real-time AI system using ReAct as a component part.

Section 8.1 -- ReAct Knowledge Sources

ReAct uses a set of five knowledge sources to provide response in real time. These are illustrated in Figure 1. In this figure small hierarchies are used to indicate the transformation on the hierarchy that each KS effects. The first thing to notice is that in this implementation there is a concept of *belief* in each particular node in the hierarchy, indicated by the amount of black shading in the diagrams. This is not a concept that has any meaning in the theoretical framework. However, because of the property mentioned above that the test outcome associated with a particular node may be noticed without the test being explicitly requested by ReAct, it is necessary to have a notion of belief in this implementation.

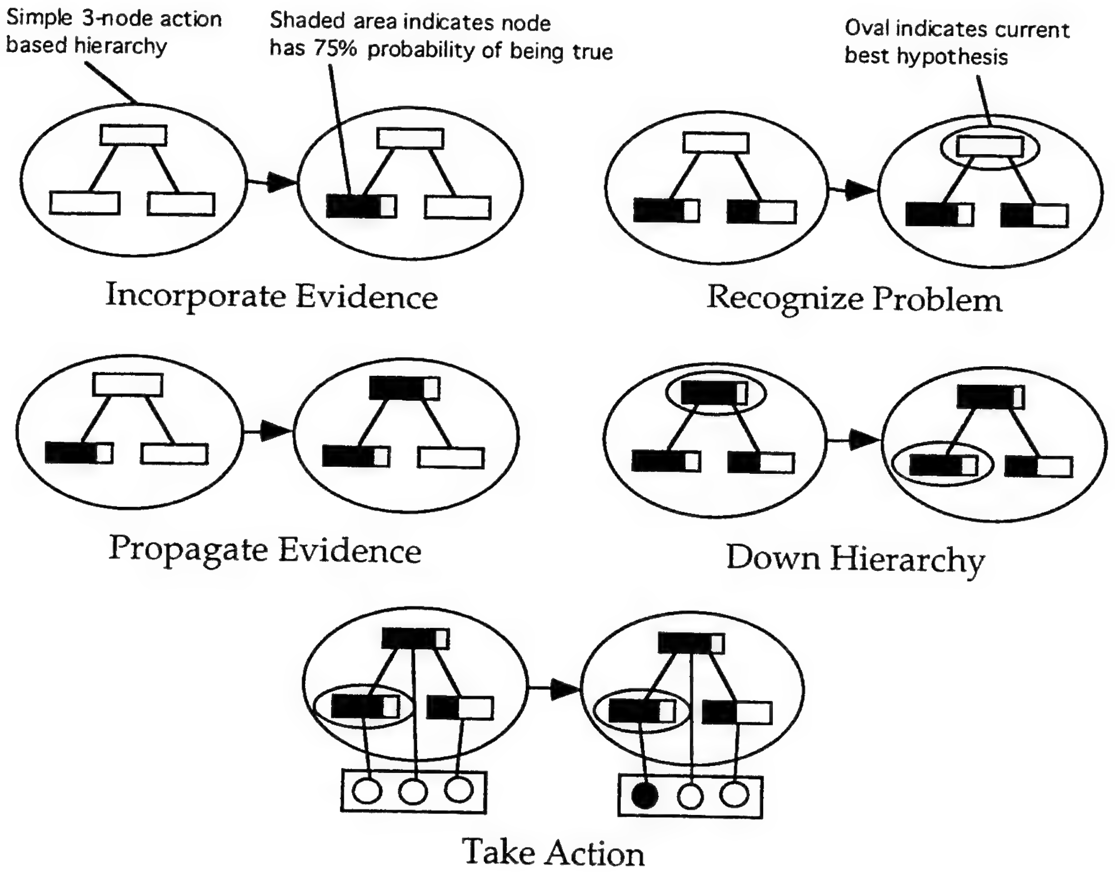

Simple 3-node action based hierarchy
Shaded area indicates node has 75% probability of being true
Oval indicates current best hypothesis
Incorporate Evidence
Recognize Problem
Propagate Evidence
Down Hierarchy
Take Action

Figure 1: ReAct Knowledge Sources

Following is a description of the meanings of each of these five knowledge sources:

Incorporate Evidence: This knowledge source causes the belief in a node in the hierarchy to change on the basis of a test outcome. Tests in ReAct are represented as a set of parameters whose values need to be obtained together with a *fuzzy pattern* describing the particular values of those parameters that give particular outcomes to the test. Note that it is not necessarily the case that the patterns representing the particular outcomes to the test are mutually exclusive. It follows that ReAct will be capable of diagnosing multiple faults using this system.

The patterns themselves are represented using the formalism associated with the fuzzy pattern recognizer (FPR) developed by Drakopoulos [DR91]. This pattern recognizer produces a level of belief in each outcome of each test, provided that all associated parameter values are available. The level of belief proposed by Drakopoulos is a possibility measure and not a probability measure, which results in it having some interesting properties. From the point of view of ReAct, the possibility measure must be reduced to a single bit representing either the outcome is present or it is not present. This will be discussed in more detail below.

In the current implementation, the patterns are non-temporal--they depend only on the current value of the parameters. However, it is possible to envision patterns that depend on the value of parameters as they vary over time. Hence, although at present the computational cost of evaluating a pattern is trivial, when the temporal variation in parameter values is added, giving a new fuzzy pattern recognizer t-FPR, the computational costs may no longer be trivial.

Recognize Problem: To understand the meaning of this KS, it is first necessary to understand what is meant by a "problem" in ReAct. If ReAct were to be used only once during any given run of the system, and were to diagnose but a single fault, then there would be no need to distinguish among different problems that ReAct might be working on. However, this is not necessarily the case. It may be that ReAct solves a problem, but later on another problem arises that requires ReAct's attention. Or it may be the case that ReAct is working on a problem, but recognizes that there is more than one fault associated with the problem and decides to split its search of the hierarchy into multiple sub-problems.

It is because of possible issues like this arising that the notion of a "problem" developed. A problem represents a single search through an action-based hierarchy, together with the history of the node believed to be the current best hypothesis at different points in the search. To this end, the Recognize Problem KS is triggered whenever there is a sufficiently high probability increase in the belief of any node in the hierarchy that is not in a part of the hierarchy covered by any existing problem that ReAct is working on. The effect of Recognize Problem is to set the reactive plan in motion.

A corollary of this is that to initiate a reactive plan in this implementation, it is necessary for there to be some means of updating belief in a node in the hierarchy without an explicit request of a test by ReAct. In this domain, it was found that the tests associated with the high level nodes in the hierarchy were, in essence, performed by Focus frequently enough that this was an effective means of initiating the ReAct diagnosis process. However, in another domain this might not be an effective means of initiating ReAct, and some other means would need to be found.

Propagate Evidence: Because it is not necessarily the case that evidence comes back to ReAct in the exact order that tests are requested (for the reasons given above) the Propagate Evidence KS is needed. The idea here is that evidence in favor of a low level node in the hierarchy may also be evidence in favor of its ancestors. The issue here is somewhat tricky, however. Evidence in favor of a low level node may simply mean that in the presence of additional evidence confirming the parent node, then this node should be activated; in may not mean anything in and of itself. It is necessary to look at the outcomes to the specific tests. If the outcome for a specific test at a specific node is a subset of the outcomes for ancestor nodes, then propagation of evidence should occur. Otherwise it clearly should not.

Down Hierarchy: This KS moves the current best hypothesis from a parent node to a child node. This should be done whenever the level of belief in the child node is sufficiently high that one would be willing to take action on the assumption that the child node is in fact present. It is an interesting question how high it is necessary for the belief level to be in order to make this determination. One possibility is to simply use a "magic number" such that a level of belief higher than some fixed threshold, say 0.65 (they range from 0 to

1) is sufficiently high to warrant action. This approach actually stands a reasonable chance of working if the question put to a domain expert when constructing the outcomes of tests is not "which outcomes would cause you to believe that this outcome is present?" but rather "which outcomes would cause you to act as though this outcome is present?" Then the possibility measure produced by FPR becomes a measure of the willingness to act, and it is reasonable that there should be a single magic number that would work.

This is the approach that has been taken in the current implementation, and it has been found to be reasonable. However, this is not a property of any particular magic number, but rather that for the particular patterns used, belief measures tend to be either very close to 0 or very close to 1 (there isn't all that much fuzziness in the fuzzy patterns that are actually being used). In other words, the work of disambiguating the semantics of the outcomes of tests has already been done, and any magic number between 0 and 1 would work except for those numbers very close to either 0 or 1. Other approaches based more on the ability to reason under uncertainty could also be taken, but these other approaches would still need to be based on the semantics that moving down the hierarchy signifies a willingness to act. Given such an approach, it would be interesting to run experiments in which the threshold is varied to determine how the results change with a variance in the threshold.

The fact that the current best hypothesis is moved down the hierarchy does not mean that there may not be faults that are descendants of sibling nodes that are also present. If this possibility is present, this KS will split off another search of the hierarchy and create a new problem in doing so.

Take Action: Corrective action must be taken when the soft deadline is reached. The question is how one recognizes that a soft deadline has been reached. In the present implementation, this is done by having a soft deadline associated with each fault in the hierarchy, and then setting the deadline for higher level nodes to be the minimum of the deadlines for the descendants. This is clearly a cruder approach than is desirable. For one thing, it can at best approximate the more complex notions of deadline distribution that are used in the analysis needed to construct the hierarchy.

Therefore, a desirable topic of future research would be one of recognizing when a deadline has been reached based on parameter values or other inputs. It is an assumption of the approach that the agent has a means of doing this, and its current methodology can be at best represent an approximation to this assumption.

Section 8.2 -- Sample ReAct Run

In Figure 2 is illustrated an simple example of a ReAct run.

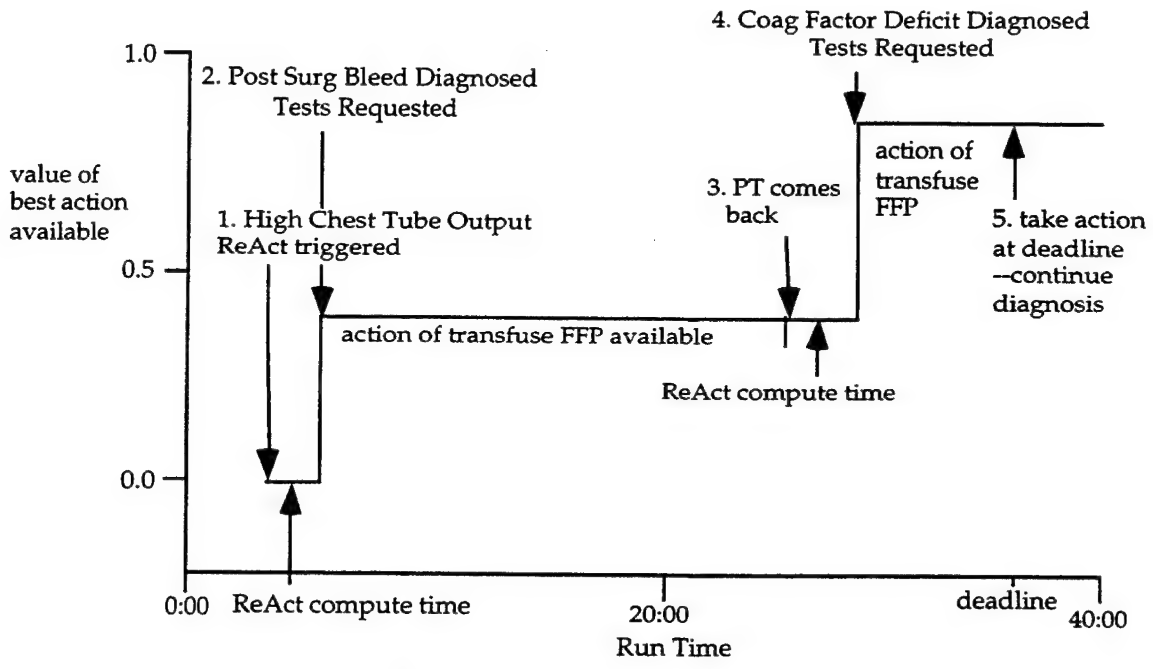


Figure 2: Sample ReAct Run

In this example the first parameter value noticed, high Chest Tube Output at step 1, is noticed outside the scope of ReAct itself (this is a requirement of this particular implementation, as mentioned earlier in this chapter). Because the agent is not working exclusively on ReAct KSes, some time elapses before the agent is able to make a refinement of the current best hypothesis, in this case diagnosing Post Surgical Bleed. Note that the agent has been able to bypass intermediate nodes in the hierarchy because this is an example of where the outcome to a test that points to a lower node also has bearing on higher level nodes, and hence it is not necessary to wait for the outcomes of those tests. The diagnosis of Post Surgical Bleed is made at step 2. The next test, a request of PT (prothrombin time) is made and comes back at step 3. Again, the agent is not exclusively working on ReAct related KSes, so some time elapses before a refined diagnosis of Coagulation Factor Deficit is made at step 4. Further tests are requested.

However, the deadline is reached at step 5, and at this point an action--transfusing fresh frozen plasma (FFP) is recommended. This does not represent the completion of the run, but further diagnosis is not shown in this diagram. Even when a deadline is reached, ReAct always continues to attempt diagnosis down to the leaf nodes of the hierarchy after recommending the action required at the deadline.

Chapter 9
Conclusions and Future Work

This thesis has illustrated both the strengths and limitations of the action-based hierarchy approach. The greatest strength of the approach is that no possible competing approach to the problem previously described in the literature is as effective at solving the problem as the action-based hierarchy approach. Many of the other technologies described in the literature do not address the problem addressed by this research; their primary drawback is that they are not simultaneously capable of providing response in real time and making the planning time/real time tradeoff so essential to this approach. The one technology that can be applied to this problem--decision trees--proves to be an effective approach, but capable of being made more so by a number of refinements which have been described in detail in this research.

The work provides an important contribution to the field in several ways. It represents an improvement on existing diagnosis work in that this work either addresses diagnosis but not from the point of view of real time constraints ([RE87], [DE87], many others), or addresses complexity issues without the specific notion of meeting a real time deadline ([CO90] and some others). With regard to existing reactive planning systems, this approach takes us beyond the limitations of having to come up with a response within a single cycle or not at all ([KA87] and others). Existing anytime systems assume deliberation (deciding what to do next) and action (doing it) are interleaved at run time ([DB88], [RU91] and others)--this approach requires that all deliberation take place at planning time. As will be discussed further below, it would be interesting to examine further the limitations and advantages this approach may offer. Existing attribute selection approaches to decision tree construction do not address the real time question, although it would be interesting to try many of them out to see whether others might do better for the real time problem than entropy--this issue has not been fully explored.

The question thus arises as to what the limitations are of this approach and whether it is reasonable to believe that further work could push back the limitations. This chapter will seek to identify a number of the more promising ideas for extension of the work; the description in this chapter is by no means intended to be complete.

Section 9.1 -- Reasoning under Uncertainty

Perhaps the most important capability that would need to be added to this approach would be the ability to reason under uncertainty. Right now there is only a limited ability in the sense that the system can handle tests that have outcomes that are consistent with more than one fault. However, there is no general ability to handle arbitrary changes in the probability of faults as the reasoning process progresses. The probability of a fault can only be zero or in the same proportion to other possible faults that existed at the beginning of the execution of the reactive plan.

It seems clear that a direct approach--planning in advance all possible combinations of probabilities of faults that may occur during the reactive planning process--would be prohibitively expensive in terms of the amount of storage space that would be consumed by the reactive plan. A separate node would be needed for each such combination, and the number of such nodes would be far greater than at present when a node is needed only for each distinct differential diagnosis.

However, the implementation described in chapter 8 provides some clues as to how reasoning under uncertainty could be incorporated into this approach. In that chapter, a fuzzy pattern recognizer is described which is able to perform reasoning under uncertainty. This is incorporated into the approach in a rather trivial way--the outcomes from the fuzzy pattern recognizer are converted into binary outcomes for the tests on the basis of whether the level of belief exceeds a particular threshold. However, this does serve as the basis for being able to reason with uncertainty. The criterion for deciding that a particular test outcome has occurred is not whether the outcome has occurred (this may be uncertain) but whether one is prepared to act as though the outcome has occurred.

Hence, there are several research issues that would need to be addressed to be able to more fully incorporate reasoning under uncertainty into the approach. The first is that a theory would need to be developed of when it is or is not appropriate to act as though a certain test outcome has occurred. Many of the criteria that are taken into account both in this thesis and in Dabija's work [DA94] would also be needed to make such a determination. For example, the costs of acting versus the costs of not acting, and the consequences and side effects of actions would need to be taken into account.

The second research problem involves making a non-trivial connection between the reasoning under uncertainty literature and this work. Approaches that could be taken include using a belief network to update belief in nodes, where the belief network is not necessarily the same structure as the action-based hierarchy--since one is used to determine level of belief and the other to determine when to act, there is no reason to believe they would be the same structure. Another possible approach would be to expand the fuzzy pattern recognizer work so that a single instance of invoking the pattern recognizer can be used to determine possibility measures for a number of nodes in the hierarchy. To successfully integrate such approaches requires that one learn how the restructuring of the hierarchy affects the shape of the belief network used to determine probabilities within the hierarchy.

Using a more sophisticated approach to reason under uncertainty requires that a more sophisticated model exist for determining the "costs" of a test. The process of performing a "test" now would involve gathering perhaps multiple pieces of information until enough information is amassed to put the belief in a particular outcome high enough that one is prepared to act. Estimating this would be an interesting challenge.

Given such an estimate, one could still use the hierarchy structuring algorithm as before, but the question arises whether the existing approach would produce a useful hierarchy in such more complex cases. This is the third research problem that has to be addressed.

Section 9.2 -- Deliberation in Real Time

The assumption underlying much of this research is that it is desirable to precompile solutions in advance to reactive problems, at least as much as possible, because the structuring process is sufficiently complex and computational resources sufficiently scarce in real time that it cannot be wasted on hierarchy structuring. Certainly we are not the only researchers to make this assumption--implicit in the work of Bratko et.al. [BR87] or Widman [WI91] or any decision tree structuring algorithm this same assumption is made. But is it a reasonable assumption to make? This is something that has not been addressed by this research.

If this assumption were not made, then rather than having a precompiled hierarchy, instead a differential diagnosis would at all times be maintained. Whenever it was needed to perform a test, one would be selected from the set of available tests according to which had the highest action-based heuristic. In other words, rather than compute the whole hierarchy in advance, only that portion of the hierarchy needed to perform diagnosis would be computed, and that portion would be computed in real time.

There are potential advantages and disadvantages to this approach. One major advantage is that concern over a possible explosion in the number of nodes would be alleviated. This is something that the complexity analysis of chapter 4 could not fully address--the empirical evidence is that it is not a problem in practice, but not having to construct the hierarchy would *guarantee* that it does not become a problem. Another advantage is that dynamic replanning would not become a costly venture from the point of view of the reactive planner. Since there in effect would no longer be a reactive planner, dynamic replanning would become essentially free. Another possible advantage would be that another approach to reasoning under uncertainty would become available. In the last section, using the action-based approach in a direct manner to reasoning under uncertainty was dismissed because of the inordinate number of nodes that it would produce. However, if the hierarchy does not need to be compiled in advance, that objection would disappear and it might become an attractive alternative.

The major disadvantage, of course, is that some computation that might be done in advance is now being done at run time. This is why it is an extremely interesting research question. It seems fairly obvious that for a sufficiently large hierarchy, it would be too costly to do this at run time, so the precompilation approach would become necessary. However, it is not clear whether the hierarchies one encounters in practice are likely to be large enough for this tradeoff to be necessary.

Another possible drawback is that it is no longer possible to do test promotion. Test promotion requires that knowledge about the structure of the hierarchy below a particular node be available. Since the structuring is being done on the fly at run time, the only information available is that directly relevant to the particular node being looked at. In certain domains, test promotion is essential and hence this is a serious drawback.

Section 9.3 -- Decay in Values of Actions

The value of an action for a particular fault is assumed to be constant up to the hard deadline, and then zero thereafter. This clearly is not valid in general. Indeed, Rutledge [RL91] has proposed that various models be used for the decay in the value of actions as a function of time.

Two issues would need to be addressed in order to handle this decay. The first would be the necessary changes to the hierarchy structuring algorithm to handle the knowledge of how values of actions decay over time. This is unlikely to be a substantially more difficult problem than the currently handled problem of deadline distributions. Whereas, at present, value is a constant in the integral that is evaluated in handling deadline distributions, in the case of decaying values it would be become a variable like the deadline distributions themselves. In fact there will no longer be any notion of a hard deadline in the theory, although the concept of a hard deadline could be implemented by having the value of an action explicitly be what it has been previously assumed to be--constant up to the hard deadline and zero afterwards.

The concept of a soft deadline also changes somewhat with this approach. It becomes possible to compute in advance what the soft deadline is based upon the information that the reactive planner has at its disposal. More precisely, it is necessary to take into account the costs of performing actions in deciding whether to perform an action at a non-terminal node. If the cost of further delay in performing the action is greater than the potential wasted cost of performing an action at a high level node that will subsequently prove to be unnecessary, then the action should be performed. The time point at which this becomes true can be computed in advance, and becomes the soft deadline. This requires that costs of actions have a more vital role in the structuring of the reactive plan. Previously they have been taken into account but only as deductions from the value of actions.

An interesting question arises as to what happens when the agent knows that it has reached a soft deadline and should act (as was previously assumed to always be the case) but has not reached the computed soft deadline--i.e. if the two notions of soft deadline conflict. The most obvious answer is that the earlier soft deadline should always be taken to be accurate, especially if the earlier soft deadline is the one the agent knows about in real time, not the previously computed soft deadline. The computed soft deadline was merely an estimate; the soft deadline known about in real time is based on more accurate information. However it would be useful to more fully investigate this question.

Probably the trickiest question to address is how to decide which decay function to use for the value of an action in the first place. To address this question, Rutledge's work is likely to be helpful.

Section 9.4 -- A Theory of Value of Action

In acquiring the necessary domain knowledge to solve this problem, it was found that providing information about the value of actions for particular faults was the most difficult problem. It was just not natural for domain experts to think in terms of value of action on a scale of 0-1; however some sort of numeric value was necessary to be able to successfully compare different values. What was more natural for domain experts to provide was qualitative information about what different actions did for different faults--completely solved the problem, solved the problem almost entirely, treated the symptoms but not the fault, etc.

Therefore, another suitable research direction to take would be to devise a theory of value of actions for faults such that domain experts would be able to provide such information more naturally. There are two main paths such research could take. The first would be to provide some mechanism for translating qualitative values into quantitative--the domain expert would provide the qualitative value while the reactive planner would use the quantitative for hierarchy structuring purposes. If the domain expert provided the translation table then the same problem would come up--the expert would not know how to assign these magic numbers. Hence it would be necessary to have a more basic theory of action such that the values could be computed without the need for the domain expert to provide them.

The other path that could be explored would be to not use quantitative measurements for values at all. Since ultimately the only purpose for the quantitative measurements is to be able to compare different actions, if another theory could be provided for doing this, quantitative measurements would no longer be necessary. The most obvious way to do this, simply ranking the different qualitative values, would probably not work. The reason is that a test that identifies a high value action for one obscure fault that is higher value than the best action identified by any other test would be recommended. This test might not be the best one to perform overall. A better way might be to also rank the faults, so that a test giving high value action for the highest ranking fault would be performed.

Section 9.5 -- Other Structuring Heuristics

It would be worthwhile to explore other possible heuristics for structuring the hierarchy. For example, the class separation heuristic of Fayyad and Irani referred to in chapter 3 would be interesting to explore more fully. Although as stated in that chapter it is unclear how to directly apply that approach because they presuppose that the goal is to identify classes of faults rather than individual faults, it is also clear that there is some connection between the two pieces of work. In practice the action-based approach succeeds because there are large classes of faults for which there is a single good (although not necessarily optimal) action to perform. It is possible that an algorithm could be developed to explicitly identify those classes of faults and then use the class separation approach to structure the hierarchy. Still, it seems that some refinement of the class separation approach would be necessary because the definition of a "class" would change as one moved down the hierarchy--when all the faults at a particular node are members of the same "class" a refined notion of class is needed at the next level.

Only slight improvements in hierarchy performance were noticed by using the deadline distribution heuristic; it would be useful to explore whether the heuristic could be refined to better take into account the information about deadline distributions. There seems to be no reason to doubt the basic hypothesis that deadline distributions affect hierarchy structuring, so it is to be expected that further explorations there would be worthwhile.

Appendix A
ReAct Programming Manual

Section A.0--Introduction and Overview

The purpose of ReAct is to provide response in real time. ReAct aims to provide optimal response by completing a diagnosis before deadline; however it will settle for a suboptimal diagnosis and action if that is the best that is possible before deadline. It does so by using an *action-based hierarchy* that is designed so as to always provide ReAct with an action to perform should it run out of time within which to make diagnoses. The higher-level nodes in the hierarchy are more general and have associated with them actions of general applicability that may nevertheless be suboptimal in specific instances; conversely, the lower-level nodes have associated actions that are very specific but optimal for their very specific purpose.

There are two main parts to ReAct which we describe in this document. The first part, known as the *hierarchy structuring algorithm*, produces a ReAct hierarchy given a set of possible faults and other information. The second part, known as the *ReAct run-time component*, uses either the hierarchy produced by the structuring algorithm or a hierarchy supplied by the user to monitor critical events in real-time and provide appropriate response.

ReAct is designed to be domain-independent; however, at present the main example of a system that actually uses ReAct is the Guardian intensive-care unit patient monitoring system. Hence this document will both use examples from Guardian and will describe how to use the particular instance of ReAct found in Guardian.

Figure 1 shows the relationship between ReAct and the necessary other parts of any system in which ReAct runs. It runs as part of the BB1 system and requires that Timeline, Focus, and FPR also be used. Mostly ReAct interacts directly with Timeline, which is a database containing all temporal data, which in a real-time system is virtually all data used by the system. ReAct works on *problems* that require response in real time, and it posts its partial solutions to Timeline. Meanwhile, data about the world originates in Focus and is sent, after being preprocessed by Focus, to Timeline. A fuzzy pattern recognizer

(FPR) notices relevant sets of parameter values and posts information about matches to Timeline. Finally, when ReAct notices that a certain parameter value may help it make further diagnoses, it may send a message to this effect directly to Focus, instructing Focus to send particular data more frequently. Readers interested in more details about these systems should consult the appropriate references for Focus [WA89], Timeline [AS90], and FPR [DR91].

Section 1 of this document describes the basic knowledge needed to run ReAct--how to load the system, and basic information about faults, actions, etc.--that is essential to understanding ReAct. Section 2 gives details on how to use the ReAct structuring algorithm to automatically generate hierarchies. It also describes the logical syntax used by ReAct to represent test outcomes and compares this syntax to that used by FPR. Finally, Section 3 describes how ReAct works when a system incorporating ReAct is actually run.

Section A.1--Basics

In this section we describe how to load ReAct as well as describe the basic information that a user of ReAct will need to have about his or her domain to make effective use of ReAct.

A.1.1 Loading BB1

ReAct runs in version 2.5 of BB1. BB1 can be loaded by loading the following file:

(load "x23:dash.demo5;bb1-setup")

When this file is loaded, the user will be given the following prompts, with the appropriate answers in boldface:

Which Guardian system? -- **Demo5**
Load which BB1 system? -- **BB1 (2.5)**
Load the BB1 communication interface? -- **Yes**
Load which BB1 system? -- **quit loading**

When this is done, the user can enter BB1 by typing **System-1**. The user will then be ready to load the system he/she is working.

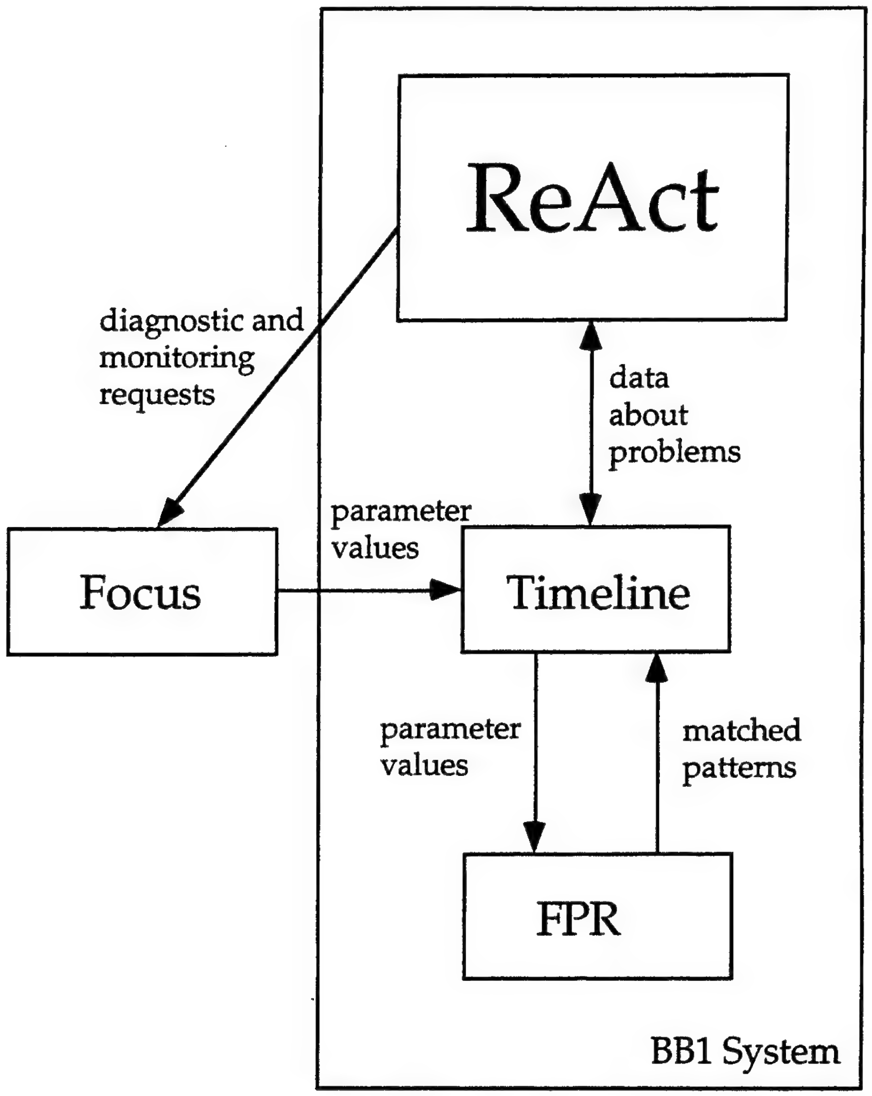


Figure 1: ReAct and necessary supporting components

A.1.2 Loading the System

Once in the BB1 screen, a system can be loaded by typing **L** for load. The user will be prompted for the system name--for the existing system, the response would be **GUARDIAN**. The user will then be asked for the file to load. The current file being used is:

x23: dash.demo5; guardian-setup

As part of the loading process, this file will load

x23: dash.demo5; demo-setup

Because the work is somewhat specific to Guardian at present, the user will be prompted to load specific Guardian subsystems. Obviously if Guardian is being used, the user will want to load many of these subsystems, but even if Guardian is not being used, the following minimal set of subsystems is required at present: **Generic, Timeline, Control-Plan, FPR, Guardian, Global-Control, Perception, ReAct.** This is because of various dependencies. It is to be hoped that as ReAct is refined, most of these dependencies will be removed, and we will be able to use ReAct after loading only **Timeline, FPR,** and **ReAct.**

The user who wishes to use ReAct as part of some system other than Guardian will probably want to modify the above files to make them appropriate for the system actually being used; however it is essential that the files associated with the above subsystems actually be loaded if ReAct is to be used successfully.

A.1.3 Files Loaded as part of ReAct

When the ReAct subsystem is loaded, the following set of files is loaded:

x23:dash.demo5;guardian-react-kb

This is the main ReAct file at present. It actually consists of two subparts: the ReAct knowledge base, which is specific to the Guardian domain, and the ReAct knowledge sources, which are generally applicable to all domains. Future versions of this file will be split into two parts.

x23:dash.demo5;react-communication-kb

This file contains a domain independent knowledge base that enables communication between ReAct and the ReAct display screen. It should always be loaded if we want to use the ReAct display.

x23:dash.demo5;bb1-fns-split

This file contains functions associated with the ReAct display that run on the BB1 machine (as opposed to the display machine).

x23:dash.demo5;react-fns

This file contains Lisp functions in support of the BB1 code in guardian-react-kb.

x23:dash.demo5;acls-algorithms

This domain-dependent file contains functions that recommend and display one of the ACLS algorithms when it is appropriate based on the results of an EKG.

A.1.4 Domain Information Required by ReAct

In this section we describe the domain knowledge that is required in order to use ReAct. It should be remembered that ReAct can be used in either of two modes--either the user supplies a hierarchy, or the user supplies a set of faults and asks ReAct to automatically structure a hierarchy based on this set of faults. Obviously in the former case, more domain knowledge will be needed in order to use ReAct; this section will make clear what body of domain knowledge is required in either case.

A.1.4.1 Faults

Faults are the most basic units of diagnosis in the ReAct system. That is, the system always endeavors to precisely diagnose a specific fault. The user of ReAct will always need to specify the set of faults that has to be diagnosed. Faults are then organized into a hierarchy of diagnoses, where each diagnosis consists of a set of faults. The organizing of faults into hierarchies of diagnoses need not be provided by the user, although it can be--if it is not provided, then the hierarchy structuring algorithm will do it for the user.

Faults are normally included on the **hierarchy.faults** level and diagnoses on the **hierarchy.diagnoses** level.

A.1.4.2 Therapeutic Actions

Also known simply as actions, therapeutic actions are actions that are taken to alleviate the faults that are diagnosed. Therapeutic actions will be performed (or recommended for performance) either when a deadline is reached or when a complete diagnosis down to a fault node is made. Normally, therapeutic actions are provided by the user, and they are found in the **react-actions.therapeutic** level. There is a **therapeutic-action** link between faults or diagnoses and their associated therapeutic actions. Either the user can provide these links, or they can be produced by the hierarchy structuring algorithm.

A.1.4.3 Monitoring and Diagnostic Actions

Also known loosely as tests, these actions are performed to gather more information about the situation when a complete diagnosis has not been made. Monitoring and diagnostic actions are similar in that they both compel the system to gather more information--the difference is that diagnostic actions cause a single parameter value to be measured once, whereas monitoring actions cause the rate at which a parameter value is measured to be increased until further notice. Monitoring and diagnostic actions are performed as necessary to complete the diagnosis process. Normally, monitoring and diagnostic actions are found on the **react-actions.monitoring** and **react-actions.diagnostic** levels respectively. There is a **monitoring-action** or **diagnostic-action** link between faults or diagnoses and their associated actions. Either the user can provide these links, or they can be produced by the hierarchy structuring algorithm. Also, there is a **comprises** link between the monitoring or diagnostic action and the specific parameter to which it corresponds.

A.1.4.4 Values

Each therapeutic action and fault pair has an associated value, which is a heuristic estimate between 0 and 1 of the value of performing the action for that specific fault. Values for therapeutic actions are not directly represented in the ReAct ontology at present in a complete fashion. However, the attribute **default-value** is used in therapeutic actions to denote the default value of that action for all faults for which it has positive value. It has positive value for a fault if it is the associated therapeutic action for that fault or any of its ancestors in the hierarchy. The default value for any action is overridden by the **specific-value** attribute, which is a list of pairs of faults and values. All value information must be provided by the user.

A.1.4.5 Costs

The cost of performing a test is indicated by the **cost** attribute of the associated parameter(s). The cost at present is simply the real time (in minutes) for performing the test. For monitoring actions, the cost is the frequency of monitoring. Note that we are using a narrow notion of cost here--the cost is simply the amount of time taken to perform the test, so especially in the case of monitoring actions, may seem to be the exact opposite of what a standard notion of cost would be. We expect to extend the notion of cost significantly in future versions of ReAct.

A.1.4.6 Priors

The prior probability of a fault is indicated by the **pp** attribute. All prior probability information must be provided by the user, and is normalized so that the sum of all priors is 1.0--ReAct makes the single fault assumption.

A.1.4.7 Tests

The notion of a *test* was introduced above as a loose equivalent of monitoring and diagnostic actions. More precisely, a test is a means for identifying whether faults are present or not present. A test has a number of outcomes, each of which is consistent only with certain faults. At present, tests are not represented in an especially precise manner. The test really consists of both the diagnostic/monitoring action, and the means of using the results of that action to refine the diagnosis. At present, the results of the action are incorporated into the hierarchy using the fuzzy pattern recognizer (FPR). Tests are needed, however, to run the hierarchy structuring algorithm that is described below.

Section A.2--Running the ReAct hierarchy structuring algorithm

ReAct provides an algorithm for automatically structuring the hierarchy. In this section we describe how to use this algorithm.

A.2.1 Loading the React hierarchy structuring algorithm

The files to load for the ReAct hierarchy structuring algorithm are:

```
(load "x12:dash.demo5;restructure")
(load "x12:dash.demo5;logical-opers")
```

The first file is termed "restructure" as opposed to simply "structure" because it is capable of taking an existing hierarchy structure and producing a new structure based upon the hierarchy structuring heuristic preferred by ReAct. This file should be loaded only after ReAct itself has been loaded as described in Section 1. The other file, "logical-opers", provides logical support for FPR-pattern related work. It is described in more detail below.

A.2.2 Running the ReAct structuring algorithm

The main function to be run in structuring the hierarchy is:

```
(restructure faults tests costs priors actions values
      &optional (heuristic #'determine-test-heuristic))
```

In the rest of this section we describe the structure of the inputs to this function:

faults -- This is a list of the faults that need to be diagnosed. ReAct assumes no internal structure whatsoever within the faults, so the elements of the list can be anything the user wants.
Example -- (f1 f2 f3)

tests -- This is an association list of test names and the partition that that test forces on the faults. Each partition, in turn, is a list as long as the number of possible outcomes of the test, with each element being the subset of the total set of faults that could be present if one of the outcomes of the test occurs.
Example --
```
(t1 ( (:faults (f1 f2)
        :result (p1 le 7.45) )
       (:faults (f3)
        :result (p1 gt 7.45) ) )
 t2 ( (:faults (f1)
        :result (p2 ge 8.12) )
       (:faults (f2 f3)
        :result (p2 lt 8.12) ) ) )
```
The **:results** of tests indicated in the above example are the specific outcomes of each test that would lead to the corresponding faults being present or not present. The syntax of these logical operations will be described in detail in Section 2.5.

costs -- This is an association list of test names and the cost (in real time consumed) of performing that test.
Example -- (t1 30.0 t2 5.0)

priors -- This is an association list of faults and the prior probabilities of each fault. Each prior must be between 0 and 1, and if they do not add to 1, the priors will be normalized.
Example -- (f1 0.7 f2 0.2 f3 0.1)

actions -- This is a list of the therapeutic actions that are available to be performed. ReAct assumes no internal structure within the actions, so they can be anything the user desires.
Example -- (a1 a2 a3)

values -- This gives the value for each (action, fault) pair. It is represented as an alist of actions and alists of faults and values.
Example -- (a1 (f1 0.0 f2 0.5 f3 1.0) a2 (f1 0.4 f2 0.4 f3 0.4) a3 (f1 1.0 f2 1.0 f3 0.1))

heuristic -- This is a function that computes, for a given test, the heuristic estimate of the effectiveness of performing that test in the diagnosis process. It is described in more detail in the following section.
Example -- determine-test-heuristic

A.2.2.1 Selecting a hierarchy structuring heuristic

As observed in the last section, the final parameter to the hierarchy structuring function, heuristic, allows the user to specify the heuristic to be used in deciding which test to perform at each level in the hierarchy. This function has the following parameters:

```
    (heuristic-fn tests-outcome-restr priors actions values faults
            test-costs value-so-far)
```

The parameters to this function are:

priors, actions, values -- Same as in the last section.

faults -- This is a subset of the total set of faults, namely it is those faults associated with the current node that are still possible diagnoses. The goal is to identify the best action to perform that will help diagnose the particular one of these faults which is actually present.

tests-outcome-restr -- This is constructed from the alist of tests (described in the last section) by first removing the cars of all the pairs, and then removing all faults that are not part of the **faults** list described above.

test-costs -- This is constructed from the alist of costs (described in the last section) by removing the cars of all the pairs.

value-so-far -- This is the expected value of the best possible action for the set of faults described above.

There are several available heuristics for structuring the hierarchy. The default heuristic, determine-test-heuristic, computes the greedy best-marginal-action function for producing an action-based hierarchy described in [AS93]. Another heuristic, **determine-test-dectree**, computes the decision-theoretic value of performing each test, again using the formula given in [AS93]. Yet a third function, included for controlling the ReAct experiments, is **determine-test-random**. This function merely assigns the next test to be performed by picking randomly from among those that help distinguish among the children of the current node.

A.2.2.2 Representation of the structured hierarchy

The hierarchy structuring algorithm implemented by the function **restructure** produces a data structure as output that is a Lisp object representing the resulting hierarchy. The structure of this Lisp object is as follows:

```
Hierarchy :=        (:result <test-result-array>
                :children <children-array>
                :test <test>
                :action-value <avg-value>
                :action <action>
                :faults <faults>)
```

Here **<children-array>** is an array with one element for each child of the current node. Each such element is a subhierarchy with the same structure as the whole hierarchy. **<test-result-array>** is an array with the same number of elements as **<children-array>**, with each element giving the test outcome corresponding to the child with the same index; the format for these outcomes will be described in section 2.5. **<test>** gives the set of parameters measured as part of the test. **<action>** gives the action with the best expected value for this node in the hierarchy. **<action-value>** gives this value. **<faults>** gives the set of faults corresponding to this node in the hierarchy.

A.2.3 Restructuring an existing hierarchy

The reader will notice that the representation for faults, etc., described in section 2.2 is distinct from that described in section 1. The representation in section 1 is a BB1 representation for complete hierarchies; the representation in section 2.2 is a Lisp representation not for complete hierarchies, but for unstructured sets of faults and their associated parameters. There is a set of functions available for converting the BB1 objects in a complete hierarchy to Lisp objects that can then be used as inputs to the restructuring algorithm. All the following functions require that the user know what the **root-node** of the hierarchy (as a BB1 object) is:

(extract-faults root-node) -- Gives the faults input to the restructuring algorithm.
(test-generation root-node) -- Gives the tests input to the restructuring algorithm.
(cost-generation (test-generation root-node))-- Gives the costs input to the restructuring algorithm.
(extract-priors (extract-faults root-node)) -- Gives the priors input to the restructuring algorithm.
(get-all-actions root-node) -- Gives the actions input to the restructuring algorithm.
(build-value-struct root-node) -- Gives the values input to the restructuring algorithm.

These functions assume that the user has a BB1 hierarchy. The user may wish to use some other representation for hierarchies. The user may do so provided that he/she provides methods for the following generic functions (methods for the **symbol** data type are already reserved for BB1 use):

(children-of node) -- Determines the children of the node.
(parent-of node) -- Determines the parent of the node.
(get-prior-prob node) -- Determines the prior probability of the node.

(assoc-pattern node) -- Determines the FPR pattern used in diagnosing a node.

(assoc-test node) -- Determines the test associated with a node. This test must be in the format of a list of the parameters that the node measures.

(get-default-value action) -- Determines the default value of an action for the set of faults for which it has positive value.

(get-specific-value action) -- Determines the specific value of an action for particular faults. Default and specific values are explained in section A.1.4.4.

(get-actions node) -- Determines the set of actions (therapeutic, monitoring, and diagnostic) associated with a node in the hierarchy.

(cost-basic test) -- Determines the cost of performing a test (in this case, the performing of a single monitoring or diagnostic action).

(get-misc-info node) -- Gets the pattern information for a specific pattern. The pattern must be in the logical format used by FPR, which is described in detail in section 2.5.

A.2.4 Building a BB1 hierarchy from a Lisp hierarchy

There is also a feature for converting in the other direction--from the Lisp hierarchy produced by the structuring algorithm to a BB1 hierarchy for use in an actual system. The function to run for this is

(build-bb1-hierarchy lisp-hierarchy)

This feature is not yet implemented.

A.2.5 Logical operations

The outcomes of tests may be represented as a logical expression based upon the values of certain parameters. The different outcomes of a particular test should satisfy the property that they are mutually exclusive but that the conjunction of all outcomes is a tautology. The outcomes of tests are used in the format for the **tests** parameter to the restructuring algorithm, and also in the **:results** pair in the hierarchy alist. The precise format for logical expressions is:

```
LogicalExpr := (and [LogicalExpr]*)
             | (or [LogicalExpr]*)
             | (not LogicalExpr)
             | GroundLiteral;

GroundLiteral := (<parameter-name> range <range>)
                 | (<parameter-name> (eq | lt | le | gt | ge)
                   <parameter-value>);
```

The semantics of these expressions should be self-explanatory. There is also a different format for logical expressions used by FPR, which the user of ReAct needs to know a little bit about because of the fact that ReAct uses FPR for pattern recognition. The FPR format for logical expressions is that returned by the **get-misc-info** function. FPR assumes that there is a list of parameters which the pattern is examining, and this list is returned by **assoc-test**. The syntax for FPR logical expressions is:

```
FPRLogExpr := (np FPRExpr);

FPRExpr := [mand | mor | and | or] ([FPRExpr | FPRLitList]*);

FPRLitList := ([FPRLitEntry]*);

FPRLitEntry := range <range>
             | [eq | lt |le | gt | ge] <parameter-value>
             | (not FPRLitEntry);
```

There is no mention of parameter names in this syntax. The parameter names are obtained from the list of parameters mentioned above. When a **mand** or **mor** is encountered, the following list of **FPRExprs** or **FPRLitLists** is applied to each element in turn of the parameter list; conversely, when an **and** or **or** is encountered, the following list is applied to the parameter or entire parameter list repeatedly.

Example: Assume the parameter list is (p1 p2).

```
(np mand (and (ge 5 le 7)) (and (le 8 ge 4)))
```

is semantically equivalent to $(5 \leq p1 \leq 7) \wedge (4 \leq p2 \leq 8)$ whereas

```
(np and (mand (ge 5 le 7)) (mand (le 8 ge 4)))
```

is semantically equivalent to $(5 \leq p1 \leq 8) \wedge (4 \leq p2 \leq 7)$

It should be noted that the above brief summary of FPR syntax and semantics by no means captures the whole of FPR and in particular omits completely the ways in which FPR handles fuzzy data, which is in fact the whole reason why FPR was designed. However, at present ReAct is not designed to deal with fuzzy data, so this section has given a brief summary of those aspects of FPR necessary to interface successfully with ReAct. A complete description of FPR may be found in [DR91].

Section A.3--Using and running ReAct

If the user wishes to design his or her own hierarchy there are essentially three files which are needed. These are the ReAct knowledge base, the FPR knowledge base, and the classify parameter file. This section provides instructions for setting up these files.

A.3.1 ReAct Knowledge Base

A sample ReAct KB is contained in the file **x12: dash.demo5; guardian-react-kb**. It is recommended that the user wishing to design his/her own ReAct KB begin with this file as a model. Within this file two types of BB1 objects are defined: diagnosis/fault objects, and actions. Here is a sample diagnosis from this file:

```
(def-bb1-object decr-contractility

  :level hierarchy.diagnoses
:attributes (
                    )

:links (
            (subdiagnosis-of
              (hierarchy.diagnoses.oligemic-hypoxia))
            (has-subfault (hierarchy.faults.ischemia
                           hierarchy.faults.myo-depr-post-cpb
                           hierarchy.faults.myo-depr-sepsis))
          (reflected-by (fpr-po.basic.decr-cont-pattern))
          (monitoring-action
            (react-actions.monitoring.map
            react-actions.monitoring.ekg
            react-actions.monitoring.hr
            react-actions.monitoring.co
            react-actions.monitoring.svr
            react-actions.monitoring.pcwp)
                    )
          (therapeutic-action
             (react-actions.therapeutic.ionotropic-agents)
                    )
          (diagnostic-action
             (react-actions.diagnostic.st-depression
              react-actions.diagnostic.chest-pain
              react-actions.diagnostic.abg)
                    )
      )
 )
```

The **subdiagnosis-of** link indicates the parent node of this node; the **has-subfault** link indicates the children nodes. Incidentally, the KB distinguishes between *faults*, which correspond to leaf nodes, and *diagnoses*, which correspond to intermediate leaf nodes (the term *diagnoses* is actually short for *partial diagnoses*). The **reflected-by** link is the link to the FPR pattern which is used to diagnose this node. The three action links all indicate associated actions of the various types.

The ReAct KB file also contains definitions of the actions known by the system. Here, from the same file, is a sample action used by ReAct:

```
(def-bb1-object increase-fio2
 :level react-actions.therapeutic
 :attributes ( (default-value 0.5)
              (specific-value ((hierarchy.faults.bronchospasm 0.6)
                               (hierarchy.faults.hypoxia 0.4)
                               (hierarchy.faults.ischemia 0.4))))
 :links nil
 )
```

Here, the **level** refers to the type of action being represented. The **default-value** is the default value of this action for all faults for which it has positive value (normally those faults which are descendants of the corresponding diagnosis in the hierarchy). The **specific-value** gives exceptions to default value.

A.3.2 FPR Knowledge Base

A sample FPR knowledge base to be used in conjunction with ReAct may be found in the file **x12: dash.demo5; fpr-pspec-kb**. Following is a sample pattern from this KB.

```
(def-bb1-object-eval normal-anion-pattern
 :level fpr-po.basic
 :attributes ((Ap #'identity_fun)
             (vp #'identity_fun)
             (np (produce_MAND
                   (produce_range 'anion/gap 'normal)
                   (produce_OR
                     (produce_range 'laba/ph 'low)
                     (produce_range 'laba/ph 'very-low))
                   (produce_range 'lab/hco3 'low)))
             (dp 3)
```

```
            (measures '(anion/gap laba/ph lab/hco3))
            (item 'normal-anion-pattern)
            (misc-info '(np MAND
                              (range normal OR
                               (range low range very-low)
                               range low)))
          )
  )
```

This pattern signifies that **anion-gap** is normal, **laba/ph** is low or very-low, and **lab/hco3** is low. A more detailed description of the semantics of FPR patterns may be found in [DR91].

A.3.3 Parameters

As has been noted a number of times earlier in this manual, ReAct uses FPR patterns in order to make diagnoses, and FPR in turn uses parameters to make diagnoses. In this section we describe how to set up parameters so that they may be used by ReAct/FPR. There is a sample file provided which shows how to set up parameters: **x12: dash.demo5; classify-parameter-kb**. There are three steps involved in adding a parameter to the ReAct system: defining the classifications for a parameter, defining the ranges, and recompiling the classifier file.

A.3.3.1 Defining the classifications for a parameter

There are two possible cases here: the parameter is something with a numerical measure, or alternatively it is a qualitative measure which takes on one of a finite, discrete set of values. In the former case, assuming the classifications of very-low, low, normal, high, and very-high are sufficient for the user's need, there will be no need to alter the classifier file. In the latter case, the user should look for the **(setf *classifier-info* ...)** form in the file, and in particular the **qualitative** subcase. Following are the qualitative classifications for the **EKG** parameter:

```
(setf *classifier-info*
  '(...
    (qualitative (...
                         (asystole 15)
                   (diastolic-disfn 16)
                   (emd 17)
                   (parox/supra/v/tach 18)
                   (v/tach/no-pulse 19)
                   (v/tach/pulse 20)
                   (vent/ectopy 21)
                   (st-segment 35) ... )
    ... )
)
```

Here the various classifications for EKG are **asystole, diastolic-disfn,** ... The numbers after the classifications have no significance other than that each classification in the system must have a unique number assigned to it. The user can add parameters to the system by following the above model. Note that in the ***classifier-info*** form, the name of the parameter itself is never used. The name of the parameter itself will appear elsewhere.

A.3.3.2 Defining the ranges for a parameter

It is also necessary to make a correspondence between numerical values for a parameter and the qualitative ranges discussed in the last section. This is done in the **(setf *parameter-info* ...)** form in the classifier file. An example of how this is done for the **EKG** parameter follows:

```
  (EKG nil
      (nil concept.natural-type.qualitative nil nil)
        (asystole diastolic-disfn emd parox/supra/v/tach
v/tach/no-pulse
            v/tach/pulse vent/ectopy st-segment)
        (nil 0 1 2 3 4 5 6 nil)
    )
```

Note that the classifications defined in the previous section are used in defining the ranges in this section. What the above does is artificially assign numerical ranges for the various qualitative values which **EKG** can have (the last line gives the artificial numerical ranges). For example, **asystole** will correspond to numerical values less than 0, **diastolic-disfn** to numerical values between 0 and 1, and so on. By using the range options for logical expressions, it will not be necessary to know these artificial values in setting up patterns. However, any simulator which produces values for qualitative parameters will need to know the artificial values.

Obviously the use of these artificial numerical values, both in this section and the previous one, is extremely awkward, and we hope that future versions of ReAct will not require that we do this.

A.3.3.3 Recompiling the classifier file

Because much of the classifier file is automatically generated and depends on parameter classifications defined above, it is necessary to regenerate and recompile the classifier file once the parameters have been defined. The code for doing so is due to Rich Washington and can be found in the file itself by searching for the string "TO CHANGE THIS FILE". Once the file has been regenerated, it will of course be necessary to recompile it.

A.3.4 Running ReAct

Once you have a complete ReAct KB, either defined by the user or generated by the hierarchy structuring algorithm, you are ready to run ReAct either stand-alone or as part of a larger BB1 system. The use of BB1 is obviously beyond the scope of this document and the interested reader is referred to [GA86]. ReAct provides a set of five knowledge sources for diagnosis and recommendation of action:

A.3.4.1 Recognize-Problem

ReAct always begins by noticing that a problem is present, normally by noticing an abnormal value of a parameter which affects some parameter associated with the FPR patterns relevant to FPR. When ReAct recognizes a problem, it posts this fact on the blackboard. As ReAct continues refining its diagnosis, it will post additional information about the problem on the blackboard. Each problem always has a *current best hypothesis*, which always starts at the top of the hierarchy and moves down the hierarchy as the hypothesis is refined.

A.3.4.2 Incorporate-Evidence

When an FPR pattern relevant to the hierarchy is noticed, the information associated with that pattern is incorporated into the hierarchy.

A.3.4.3 Down-Hierarchy

This KS changes the current best hypothesis for one of the problems that ReAct is working on from a parent node to one of its children. At the same time, a decision is made as to whether all alternative children have been ruled out. If so, no new problems will be created. If not, an alternative problem will be posted to the timeline because the possibility is that more than one fault is present. ReAct will then be working on two or more problems at once.

A.3.4.4 Recommend-Action

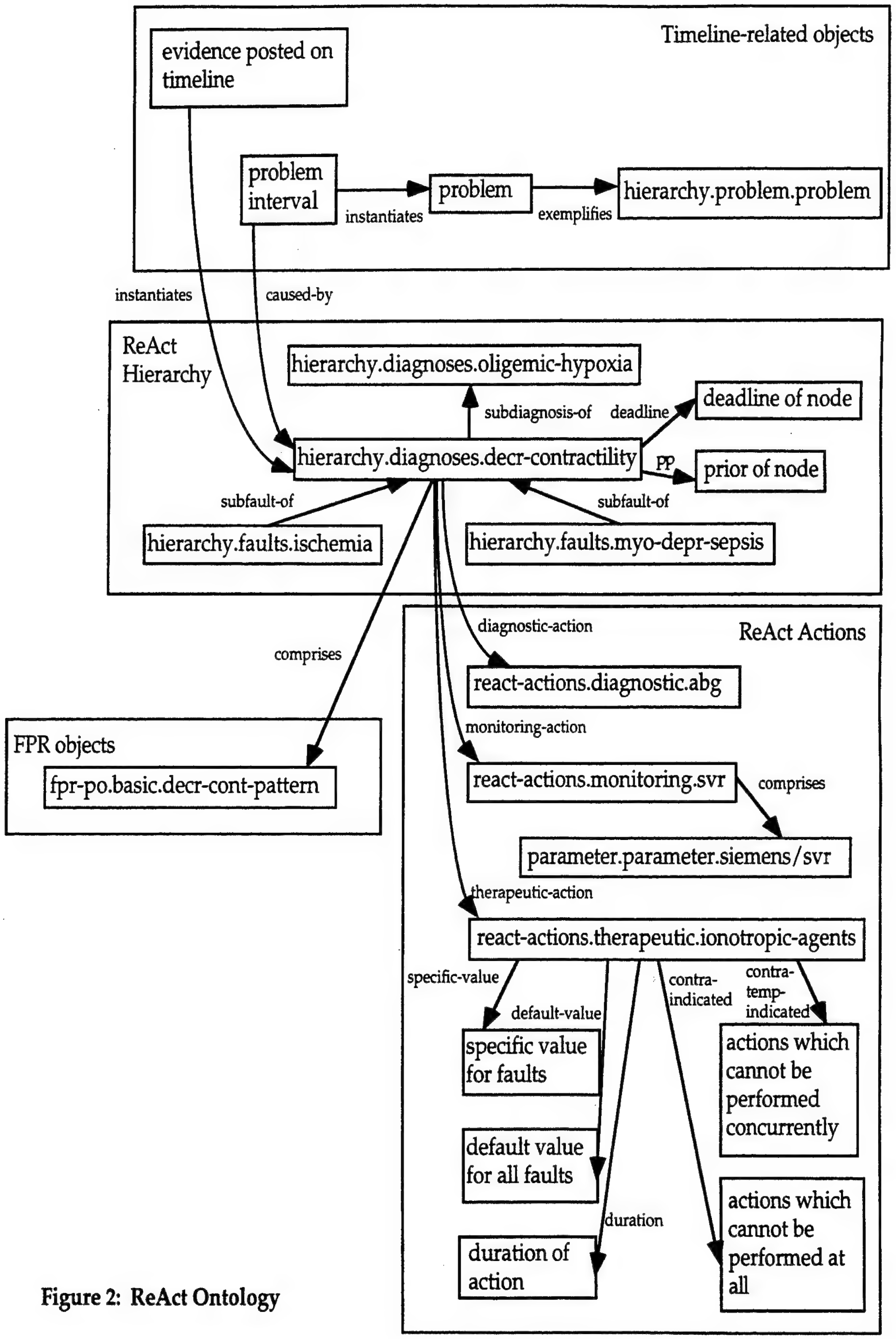
This KS activates when either a deadline is reached or a leaf node of the hierarchy becomes the current best hypothesis. The effect of this KS is to recommend the therapeutic action associated with the current best hypothesis. If we are not at a leaf node, however, we do not stop diagnosis just because we have recommended an action.

A.3.4.5 Coordinate-Actions

Because ReAct does not make the single fault assumption, it is possible that it will recommend therapeutic actions associated with more than one node at a time. There is the possibility that there will be conflicts between these actions in that they cannot both be performed at the same time. This KS looks at the potential conflicts and attempts to resolve them. For actions, the links **contra-indicated** and **contra-temp-indicated** are used to indicate those other actions which either cannot be performed if this one is performed, or cannot be performed at the same time. This KS attempts to resolve this conflict by constructing a schedule of actions to perform.

A.3.5 ReAct Ontology

Figure 2 shows the complete ReAct ontology with all links that are relevant to what has been discussed above.



Timeline-related objects
evidence posted on timeline
problem interval
instantiates
problem
exemplifies
hierarchy.problem.problem
instantiates
caused-by
ReAct Hierarchy
hierarchy.diagnoses.oligemic-hypoxia
subdiagnosis-of
deadline
deadline of node
hierarchy.diagnoses.decr-contractility
PP
prior of node
subfault-of
subfault-of
hierarchy.faults.ischemia
hierarchy.faults.myo-depr-sepsis
comprises
diagnostic-action
ReAct Actions
react-actions.diagnostic.abg
monitoring-action
FPR objects
fpr-po.basic.decr-cont-pattern
react-actions.monitoring.svr
comprises
parameter.parameter.siemens/svr
therapeutic-action
react-actions.therapeutic.ionotropic-agents
specific-value
default-value
contra-indicated
contra-temp-indicated
specific value for faults
actions which cannot be performed concurrently
default value for all faults
duration
duration of action
actions which cannot be performed at all

Figure 2: ReAct Ontology

Appendix B
Medical Knowledge Base

In this appendix is given excerpts from the medical knowledge base. This KB was prepared by our medical colleague Dr. Garry Gold with input from Drs. David Gaba and Adam Seiver.

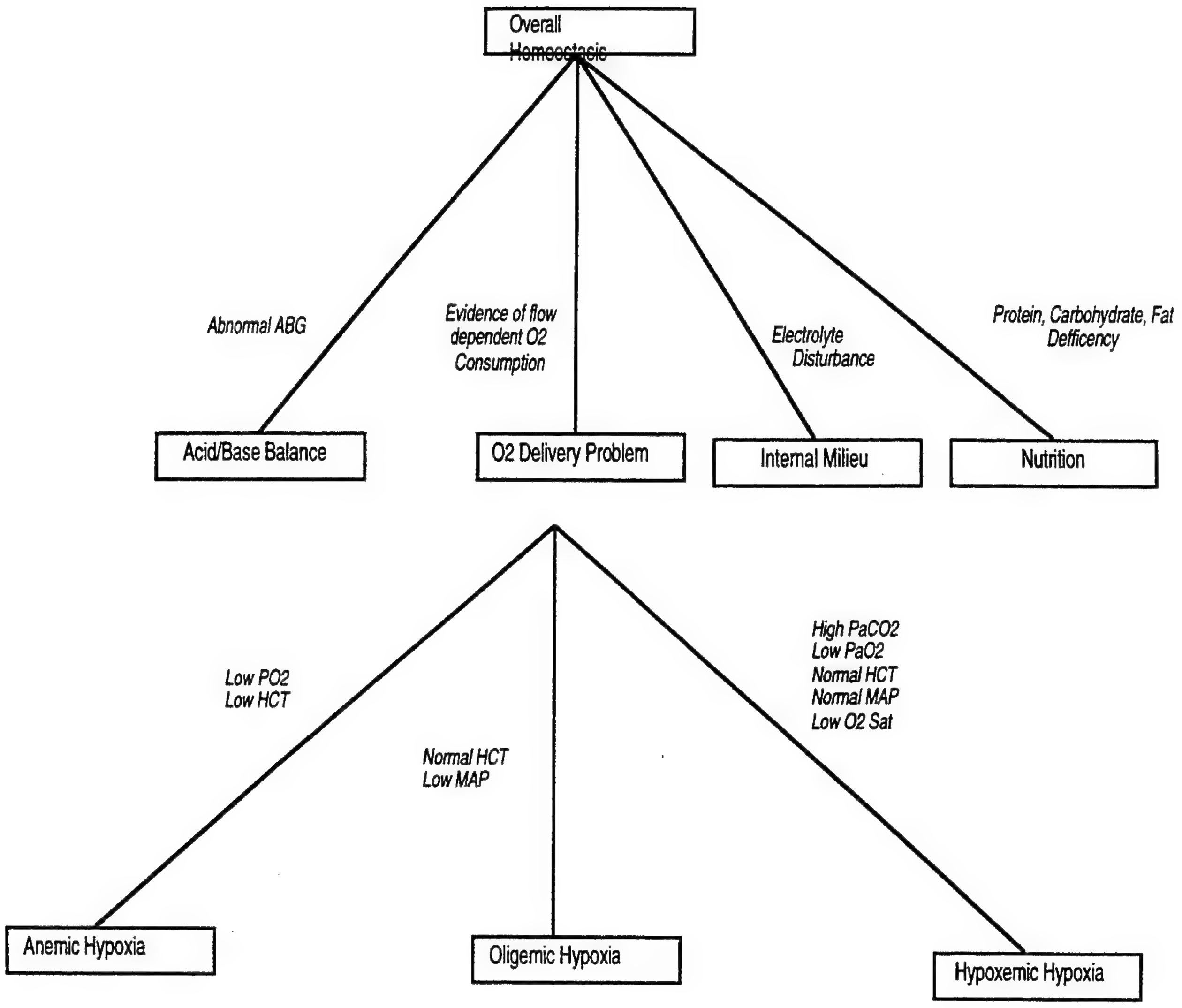
It is important to realize that this appendix represents the medical knowledge base as provided by the medical experts and *not* the inputs to the hierarchy structuring algorithm. For one thing, the medical KB was provided in the form of a hierarchy, but this hierarchy should not be taken to be an action-based hierarchy: it turned out that this form was a convenient way to provide medical knowledge, but it was not assumed that the knowledge as provided by a medical expert would take precisely the same form as that constructed by the hierarchy structuring algorithm.

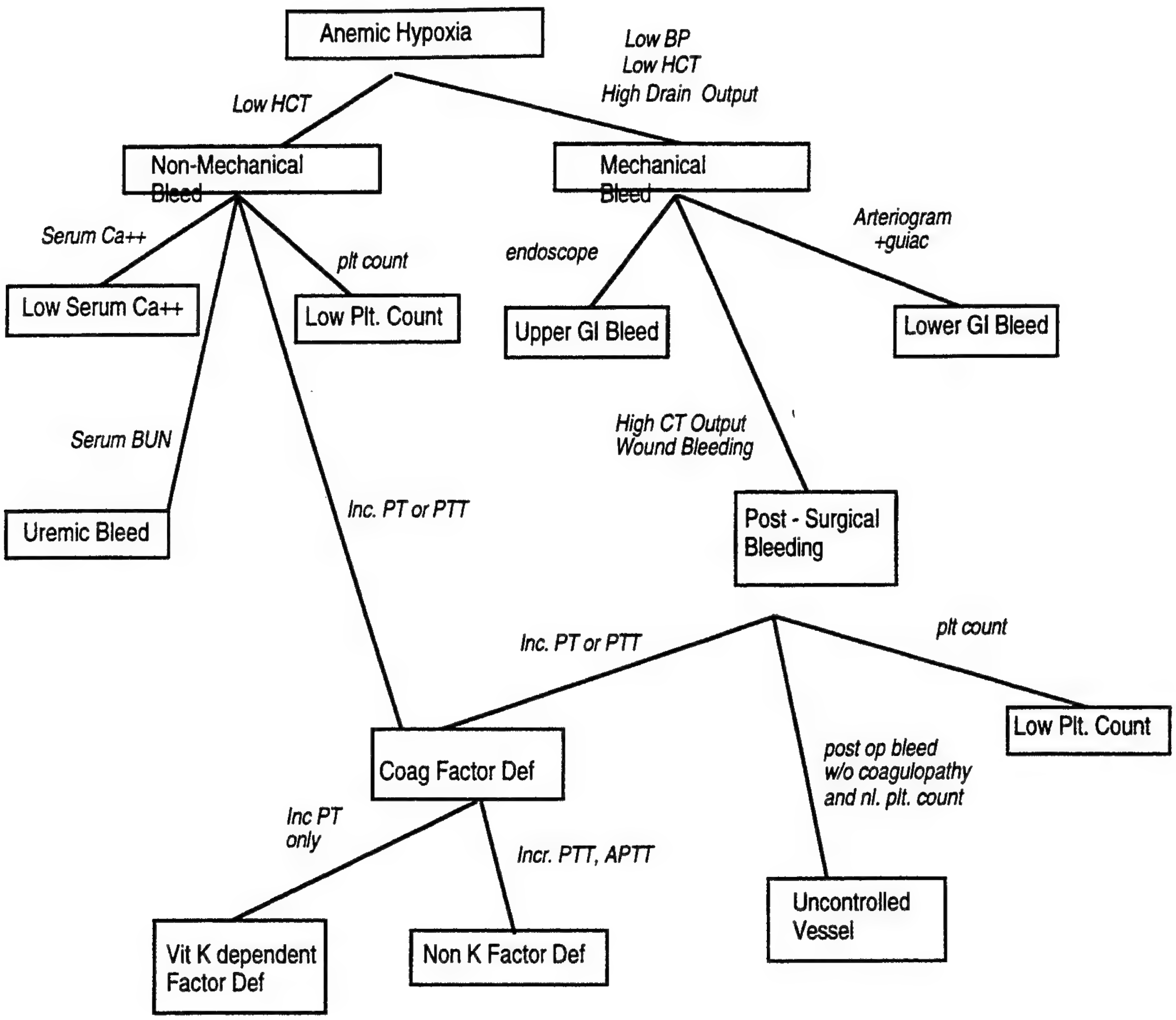
Thus, there was still a significant amount of work involved in translating the medical KB to a form which could be used for this study. In some cases this involved simplification--where more than one corrective action was given for a single leaf fault, for example. In other cases, further amplification was sought from domain experts--for example, where it was needed to know precisely what information the various tests provided about faults. This latter was actually a fairly complex process because the notion of a test is not accurately represented in the medical KB--it was necessary to extract this information. For example, different parts of the hierarchy might rely on the same set of parameters--in this case it was necessary to notice that they really represented a single test.

The medical KB is presented in the form given by the medical experts for a couple of reasons: first, it has some structure which would be lacking in the actual inputs to the structuring algorithm, meaning those inputs would be very hard to comprehend, and second, in cases where simplifications needed to be made, the reader may be interested in solving the more complete problem implied by the medical KB.

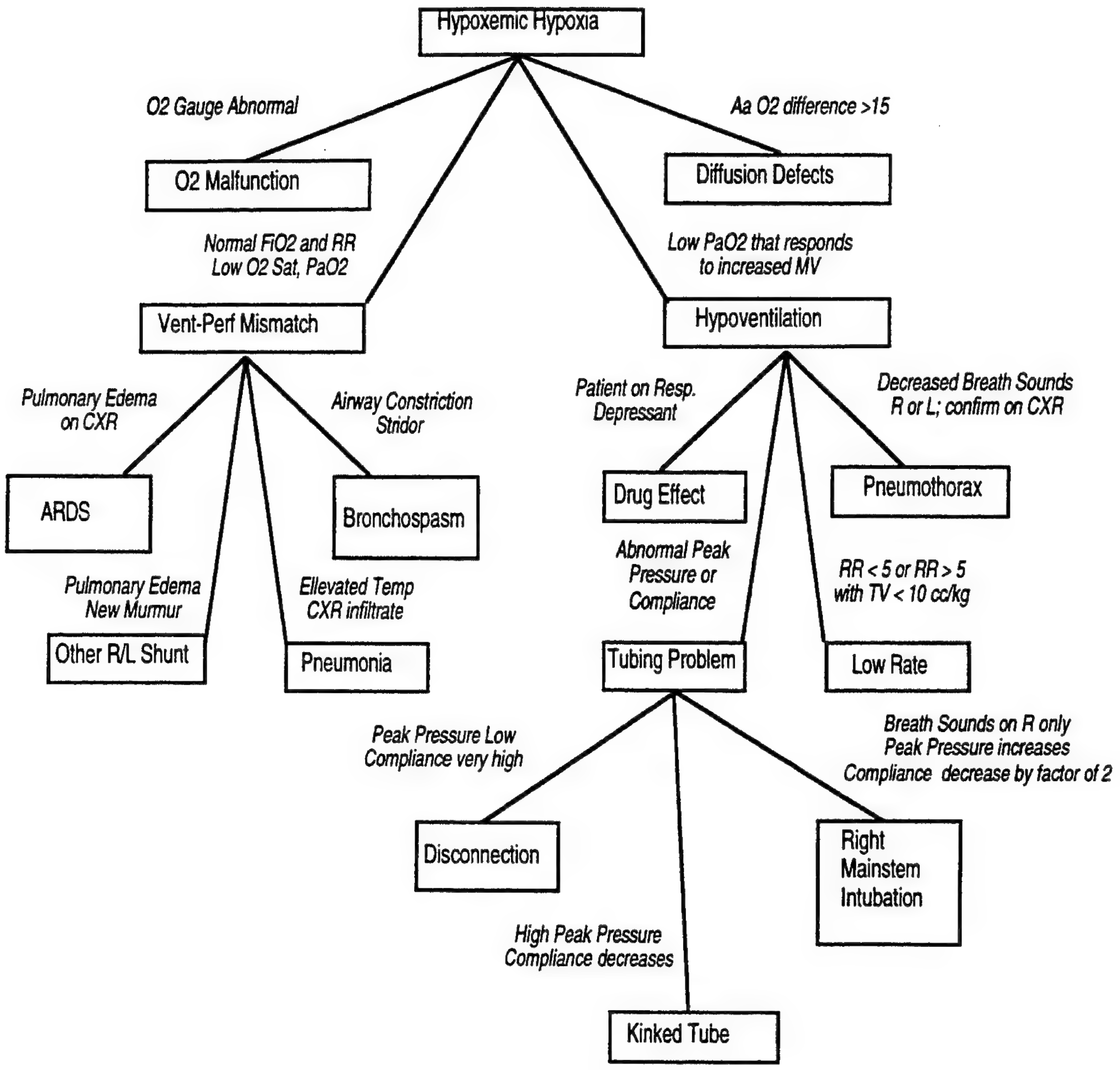
In particular, the monitoring and diagnostic actions given here are completely ignored by the structuring algorithm, because it is part of the goal of the structuring algorithm to *determine* when monitoring and diagnostic actions are required. However, the presence of this knowledge inspired the work on test promotion described in this thesis.

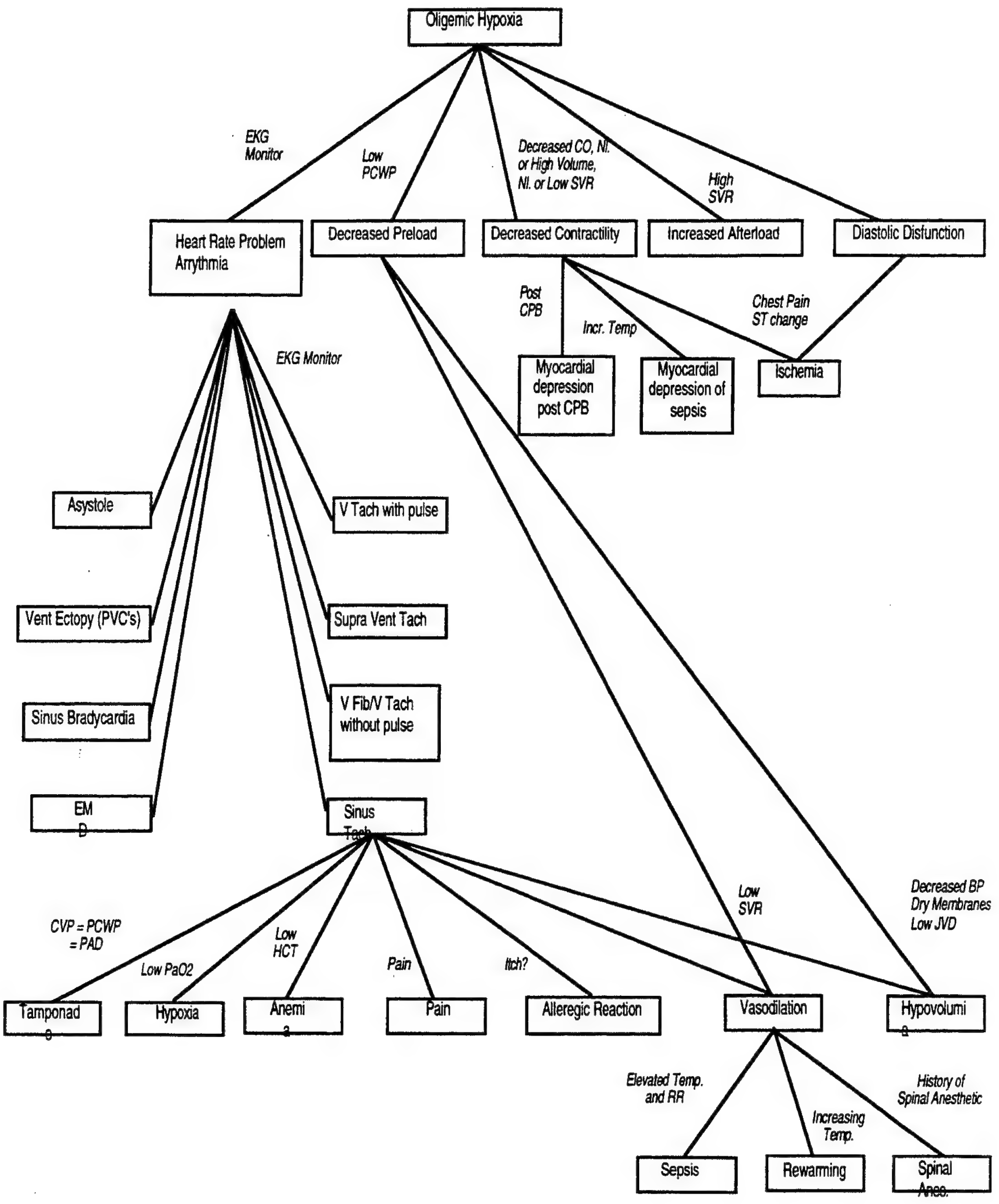
The first part of this appendix shows all the nodes in the hierarchy provided by the medical experts. The second part of the appendix shows detailed information for a part of the hierarchy referred to as "anemic hypoxia".


Overall Homeostasis
Abnormal ABG
Evidence of flow dependent O2 Consumption
Electrolyte Disturbance
Protein, Carbohydrate, Fat Defficency
Acid/Base Balance
O2 Delivery Problem
Internal Milieu
Nutrition
Low PO2
Low HCT
Normal HCT
Low MAP
High PaCO2
Low PaO2
Normal HCT
Normal MAP
Low O2 Sat
Anemic Hypoxia
Oligemic Hypoxia
Hypoxemic Hypoxia

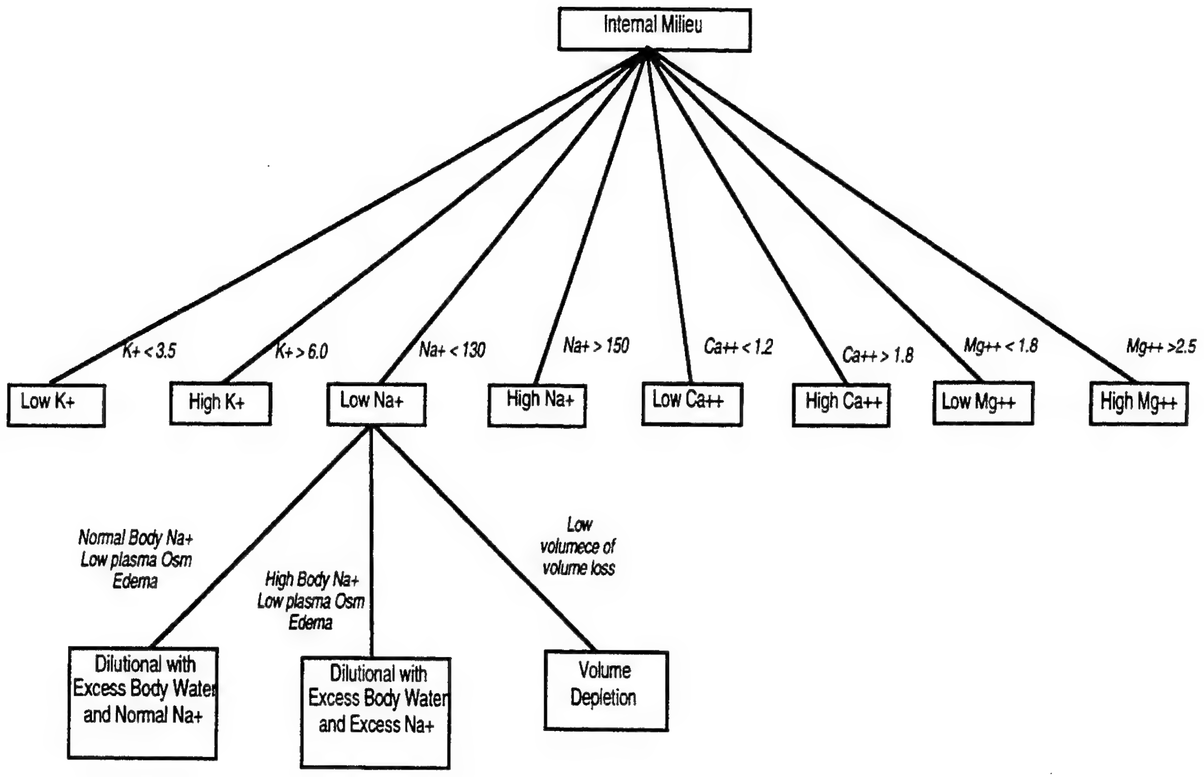


Anemic Hypoxia
Low HCT
Low BP
Low HCT
High Drain Output
Non-Mechanical Bleed
Mechanical Bleed
Serum Ca++
plt count
endoscope
Arteriogram +guiac
Low Serum Ca++
Low Plt. Count
Upper GI Bleed
Lower GI Bleed
Serum BUN
High CT Output
Wound Bleeding
Uremic Bleed
Inc. PT or PTT
Post - Surgical Bleeding
Inc. PT or PTT
plt count
Low Plt. Count
Coag Factor Def
post op bleed w/o coagulopathy and nl. plt. count
Inc PT only
Incr. PTT, APTT
Vit K dependent Factor Def
Non K Factor Def
Uncontrolled Vessel

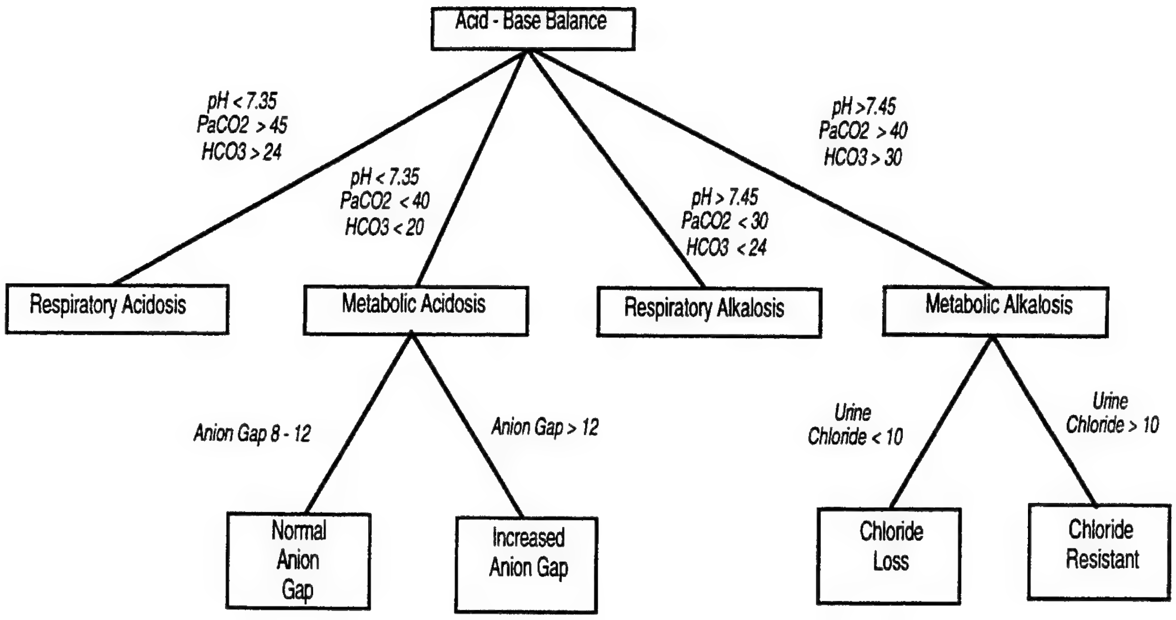

Hypoxemic Hypoxia
O2 Gauge Abnormal
O2 Malfunction
Aa O2 difference >15
Diffusion Defects
Normal FiO2 and RR Low O2 Sat, PaO2
Vent-Perf Mismatch
Low PaO2 that responds to increased MV
Hypoventilation
Pulmonary Edema on CXR
ARDS
Airway Constriction Stridor
Bronchospasm
Pulmonary Edema New Murmur
Other R/L Shunt
Elevated Temp CXR infiltrate
Pneumonia
Patient on Resp. Depressant
Drug Effect
Decreased Breath Sounds R or L; confirm on CXR
Pneumothorax
Abnormal Peak Pressure or Compliance
Tubing Problem
RR < 5 or RR > 5 with TV < 10 cc/kg
Low Rate
Peak Pressure Low Compliance very high
Disconnection
Breath Sounds on R only Peak Pressure increases Compliance decrease by factor of 2
Right Mainstem Intubation
High Peak Pressure Compliance decreases
Kinked Tube



Oligemic Hypoxia
EKG Monitor
Low PCWP
Decreased CO, Nl. or High Volume, Nl. or Low SVR
High SVR
Heart Rate Problem Arrythmia
Decreased Preload
Decreased Contractility
Increased Afterload
Diastolic Disfunction
Post CPB
Incr. Temp
Chest Pain ST change
Myocardial depression post CPB
Myocardial depression of sepsis
Ischemia
EKG Monitor
Asystole
V Tach with pulse
Vent Ectopy (PVC's)
Supra Vent Tach
Sinus Bradycardia
V Fib/V Tach without pulse
EMD
Sinus Tach
Low SVR
Decreased BP Dry Membranes Low JVD
CVP = PCWP = PAD
Low PaO2
Low HCT
Pain
Itch?
Tamponade
Hypoxia
Anemia
Pain
Allergic Reaction
Vasodilation
Hypovolumia
Elevated Temp. and RR
Increasing Temp.
History of Spinal Anesthetic
Sepsis
Rewarming
Spinal Anes.



Internal Milieu
K+ < 3.5
K+ > 6.0
Na+ < 130
Na+ > 150
Ca++ < 1.2
Ca++ > 1.8
Mg++ < 1.8
Mg++ >2.5
Low K+
High K+
Low Na+
High Na+
Low Ca++
High Ca++
Low Mg++
High Mg++
Normal Body Na+ Low plasma Osm Edema
High Body Na+ Low plasma Osm Edema
Low volumece of volume loss
Dilutional with Excess Body Water and Normal Na+
Dilutional with Excess Body Water and Excess Na+
Volume Depletion


Acid - Base Balance
pH < 7.35
PaCO2 > 45
HCO3 > 24
pH < 7.35
PaCO2 < 40
HCO3 < 20
pH > 7.45
PaCO2 < 30
HCO3 < 24
pH >7.45
PaCO2 > 40
HCO3 > 30
Respiratory Acidosis
Metabolic Acidosis
Respiratory Alkalosis
Metabolic Alkalosis
Anion Gap 8 - 12
Anion Gap > 12
Urine Chloride < 10
Urine Chloride > 10
Normal Anion Gap
Increased Anion Gap
Chloride Loss
Chloride Resistant

Anemic Hypoxia

Node: O2 Delivery Problem (pp = .6, more)
How Diagnosed: Evidence of flow-dependent O2 consumption (linear relationship between VO2 and DO2), High Lactate levels, Acidosis, Urine output < .5 cc/kg/hr, O2 sat <95, HR >110
Differential Diagnosis: Anemic Hypoxia, Oligemic Hypoxia, Hypoxemic Hypoxia
Monitoring Actions: Monitor VO2, VO2/DO2 relationship, MAP, EKG monitor, O2 Sat, PCWP, PAD, CVP, SVR, RR, ET CO2.
Therapeutic Actions: None
Diagnostic Actions: **1.** ABG (pH, pO2, pCO2) (20 min). **2.** Cardiac Output (5 min). **3.** HCT (5 min). **4.** Ask patient about chest pain if conscious (2 min) (to rule out ischemia). **5.** Order 12 lead EKG (20 min) (ischemia). **6.** Order CXR (30 min) (pneumothorax, pneumonia).

Node: Anemic Hypoxia (pp = .4, less)
How Diagnosed: Low HCT and Low pO2 on ABG with normal MAP, PCWP, SVR. ET CO2 high.
Differential Diagnosis: Mechanical Bleed, Non-mechanical Bleed
Monitoring Actions: Increase monitoring of HCT, MAP, CT output, O2 Sat, PCWP, PAD, CVP. Monitor MAP and PCWP, PAD, CVP for signs of increase and possible volume overload, which can be treated with diuretics (Lasix 10-20mg IV).
Therapeutic Actions: **1.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75).
Diagnostic Actions: **2.** Monitor pO2 regularly with ABG (20 min). **3.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **4.** Monitor temperature for hypothermia (true for any transfusion >4 units). **5.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min) **6.** Also send K+. (10 min) **7.** Monitor for possible transfusion reactions. (10 min) **8.** If GI source suspected by history, do stool guiac (10 min)

Node: Mechanical Bleed (pp = .3, less)
How Diagnosed: Low MAP, Low HCT, High CT output or blood from GI tract. MAP can fall very rapidly.
Differential Diagnosis: Upper GI Bleed, Lower GI Bleed, Post-Surgical Bleed
Monitoring Actions: Increase monitoring of HCT, MAP, CT output, O2 Sat, PCWP, PAD, CVP. Monitor MAP and PCWP, PAD, CVP for signs of increase and possible volume overload, which can be treated with diuretics (Lasix 10-20mg IV).
Therapeutic Actions: **1.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75).
Diagnostic Actions: **2.** Monitor pO2 regularly with ABG (20 min). **3.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **4.** Monitor

temperature for hypothermia (true for any transfusion >4 units). **5.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min) **6.** Also send K+. (10 min) **7.** Monitor for possible transfusion reactions. (10 min) **8.** If GI source suspected by history, do stool guiac (10 min).

Node: Post-Surgical Bleeding (pp = .15, less)
How Diagnosed: Low MAP, Low HCT, High CT output, or CT output that does not decrease over time. MAP can fall very rapidly.
Differential Diagnosis: Coagulation Factor deficit, Uncontrolled vessel, Low Platelet count
Monitoring Actions: Increase monitoring of HCT, MAP, CT output, O2 Sat, PCWP, PAD, CVP. Monitor MAP and PCWP, PAD, CVP for signs of increase and possible volume overload, which can be treated with diuretics (Lasix 10-20mg IV).
Therapeutic Actions: 1. Transfuse 6 units of platelets and 500 units of FFP (treatment effective for all lower nodes but not necessarily definitive therapy, value = .85). **2.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75).
Diagnostic Actions: **3.** Check for blood from vomit or rectum, melanic stools (5 min). **4.** Monitor pO2 regularly with ABG (20 min). **5.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **6.** Monitor temperature for hypothermia (true for any transfusion >4 units). **7.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min) **8.** Also send K+. (10 min) **9.** Monitor for possible transfusion reactions. (10 min) **10.** Stool Guiac (10 min).

Node: Lower GI Bleed (pp = .15, less)
How Diagnosed: Copious bright red blood from rectum; +stool guaic, +arteriogram or red cell scan.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT out, Rectal output, PCWP, PAD, CVP.
Therapeutic Actions: 1. Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75). **2.** Prepare patient for surgery if source can be identified (definitive therapy, value = 1.0).
Diagnostic Actions: **3.** Order red cell scan (2 hr) or arteriogram (2 hr)to find source of bleed; endoscopy (1 hr) to rule out profuse upper GI bleed. **4.** Frequent HCT (5 min) and ABG (20 min) if bleeding continues. **5.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **6.** Monitor temperature for hypothermia (true for any transfusion >4 units). **7.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min). **8.** Also send K+ (10 min). **9.** Monitor for possible transfusion reactions (10 min)

Node: Upper GI Bleed (pp = .2, less)
How Diagnosed: Low HCT and acute drop in MAP and/or bright red blood per rectum if acute, dark stools (melena) if slower. Usually diagnosed by endoscopy (fiber optic scope put in stomach).

Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT out, Rectal output, PCWP, PAD, CVP.
Therapeutic Actions: **1.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy (value = .75)). **2.** Acute treatment is usually sclerosis of the bleeding vessel via endoscopy or by surgery (definitive therapy, value = 1.0). *Prevention is usually done thru a combination of maintenance of adequate blood volume, oxygenation, and cardiac function. H2 blockers which prevent stomach acid secretion (Cimetidine and Ranitidine) or Sucralfate also help prevent ulcers. Infection control, and prevention of coagulopathy (inability of blood to clot) is also important.*
Diagnostic Actions: **3.** Order endoscopy (1 hr) to confirm upper GI bleed. **4.** Frequent HCT (5 min) and ABG (20 min) if bleeding continues. **5.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **6.** Monitor temperature for hypothermia (true for any transfusion >4 units). **7.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min). **8.** Also send K+ (10 min). **9.** Monitor for possible transfusion reactions (10 min)

Node: Uncontrolled Vessel (pp = .15, less)
How Diagnosed: Low HCT and acute drop in MAP accompanied by large increase in CT output.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT out, PCWP, PAD, CVP.
Therapeutic Actions: **1.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75). Continue to transfuse and wait for evidence bleed is slowing down. **2.** May need surgical therapy if bleeding at high rate CT > 300 cc/hr or prolonged > 5 hr (definitive therapy, value = 1.0).
Diagnostic Actions: **3.** Frequent HCT (5 min) and ABG (20 min) if bleeding continues. **4.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **5.** Monitor temperature for hypothermia (true for any transfusion >4 units). **6.** Send serum Ca++ after infusion (possible citrate toxicity) (30 min). **7.** Also send K+ (10 min). **8.** Monitor for possible transfusion reactions (10 min)

Node: Non-mechanical Bleeding (pp = .10, less)
How Diagnosed: Anemic hypoxia (low HCT and low pO2) with normal MAP and PCWP, CVP, PAD and without obvious bleeding or excessive CT output.
Differential Diagnosis: Low serum Ca++, Low Plt. Count, Uremic Bleeding, Coagulation Factor Deficit
Monitoring Actions: Increase monitoring of MAP, CT output, O2 Sat, PCWP, PAD, CVP. Monitor temperature for possible infection.

Therapeutic Actions: **1.** Transfuse 1 unit of packed RBC's per every 2% the patient is below normal HCT (treatment effective for all lower nodes but not necessarily definitive therapy, value = .75).
Diagnostic Actions: **2.** Frequent HCT (5 min) and ABG(20 min) if bleeding continues. **3.** Send for platelet levels (30 min), Coagulation factor levels (1 hr), serum BUN (30 min), DIC Screen, BT, TT (all 1 hr), PT, PTT (30 min). **4.** If infection is suspected, send Blood cultures (24 hrs) and begin empiric antibiotics. **5.** Monitor temperature for hypothermia (true for any transfusion >4 units). **6.** Send serum Ca++ after infusion (30 min) (possible citrate toxicity). **7.** Also send K+ (10 min). **8.** Monitor for possible transfusion reactions (10 min).

Node: Coagulation Factor Deficiency (pp = .05, less)
How Diagnosed: PT (prothrombin time) is elevated in the case of vitamin K dependent factor deficiency. In liver disease PT, APTT(activated partial thromboplastin time) and TT (thrombin time) are all elevated.
Differential Diagnosis: Could be due to deficiency in either Vitamin K dependent factors of non-Vitamin K dependent factors
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: **1.** Transfuse 500 cc FFP and 120 cc platelets (nearly definitive therapy, value = .95).
Diagnostic Actions: **2.** Determine if the coagulation factors are Vitamin K dependent factors or not by PT, PTT (30 min). **3.** Monitor for possible transfusion reactions (10 min). **4.** Check LFT's for possible liver disease if not already done (4 hrs).

Node: Vitamin K Dependent Coagulation Factor Deficiency (pp = .03, more)
How Diagnosed: PT (prothrombin time) is elevated in the case of vitamin K dependent factor deficiency.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: **1.** Acutely transfuse 500 cc FFP and 120 cc platelets (definitive therapy - short term, value = .95). **2.** IV Vitamin K (definitive therapy - long term, value = 1.0).
Diagnostic Actions: **3.** Monitor for possible transfusion reactions (10 min). **4.** Check LFT's for possible liver disease if not already done (4 hrs). *The syndrome known as DIC (disseminated intrvascular coagulation) often presents with elevated PT, low platelets, and high serum BUN together in a critical patient.*

Node: Non-Vitamin K dependent Coagulation Factor Deficiency (pp = .01, less)
How Diagnosed: PTT (activated partial thromboplastin time) is elevated and PT is normal.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: **1.** Transfuse 500 cc FFP and 120 cc platelets (definitive therapy, value = 1.0).
Diagnostic Actions: **2.** Monitor for possible transfusion reactions (10 min). **3.** Check LFT's for possible liver disease if not already done (4 hrs).

Node: Low Platelet Count (pp = .1, more)
How Diagnosed: CBC with manual platelet count of <100,000 per ul or low level of functioning platelets due to post-CPB state.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: 1. Transfuse 120 cc platelets (definitive therapy, value = 1.0).
Diagnostic Actions: 2. Monitor for possible transfusion reactions (10 min).

Node: Low Serum Calcium (pp = .05, same)
How Diagnosed: Ionized serum Ca++ less than 1.2 mg/dl. Bleeding or low HCT with normal serum BUN and normal Plt count, PT, PTT.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of EKG Monitor, MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: 1. Intravenous Calcium Gluconate (definitive therapy, value = 1.0).
Diagnostic Actions: 2. Monitor for evidence of muscle twitches (2 hrs). **3.** Send serum Ca++ (30 min).

Node: Uremic Bleeding (p = .05, more)
How Diagnosed: Bleeding or low HCT with elevated serum BUN and normal Plt count, PT, PTT.
Differential Diagnosis: None
Monitoring Actions: Increase monitoring of MAP, O2 Sat, CT output, PCWP, PAD, CVP.
Therapeutic Actions: 1.120 cc platelets and dDAVP (buying time, value = .6). **2.** Consider dialysis (definitive therapy, value = 1.0).
Diagnostic Actions: 3. Monitor mental status for evidence of coma (10 min). **4.** Monitor serum electrolytes (30 min). **5.** Send LFT's to check liver function (4 hrs).

References

[AS90] D. Ash and B. Hayes-Roth, Temporal representations in blackboard architectures, Technical Report KSL-90-16, Knowledge Systems Laboratory, Stanford University (1990).

[AS93] D. Ash and B. Hayes-Roth, A comparison of action-based hierarchies and decision trees for real-time performance, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (1993) 568-573.

[AH93] D. Ash, G. Gold, A. Seiver and B. Hayes-Roth, Guaranteeing real-time response with limited resources, *Artificial Intelligence in Medicine* 5 (1993) 49-66.

[BR87] I. Bratko, I. Mozetic and N. Lavrac, Automatic synthesis and compression of cardiological knowledge, *Machine Intelligence* 11 (1987) 435-454.

[BR84] L. Breiman, J. Friedman, R. Olshen and C. Stone. *Classification and Regression Trees*, Monterey, Calif.: Wadsworth & Brooks (1984).

[BR86] R. Brooks. A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* (March, 1986) 14-23.

[CH88] J. Cheng, U. Fayyad, K. Irani and Z. Qian, Improved decision trees: a generalized version of ID3, *Proceedings of the Fifth International Conference on Machine Learning* (1988) 100-108.

[CH91] L. Chrisman and R. Simmons, Sensible planning: focusing perceptual attention, *Proceedings of the Ninth National Conference on Artificial Intelligence* (1991) 756-761.

[CL85] W. Clancey, Heuristic classification, *Artificial Intelligence* 27 (1985) 289-350.

[CN89] P. Clark and T. Niblett, The CN2 induction algorithm, *Machine Learning* 3 (1989) 261-284.

[CL89] J. Clarke, M. Niv, B. Webber, K. Fisherkeller, D. Southerland and B. Ryack, TraumAID: a decision aid for managing trauma at various levels of resources, *Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care*, Washington, DC (1989).

[CO90] G. Cooper, The computational complexity of probabilistic inference on Bayesian belief networks, *Artificial Intelligence* 42 (1990) 353-405.

[DA94] V. Dabija, Deciding whether to plan to react, Ph.D. dissertation, Computer Science Department, Stanford University (1994).

[DA88] R. Davis and W. Hamscher, Model-based reasoning: troubleshooting, *Exploring Artificial Intelligence: Survey Talks from the National Conference on Artificial Intelligence*, ed. H. Shrobe (1988) 297-346.

[DE87] J. De Kleer and B. Williams, Diagnosing multiple faults, *Artificial Intelligence* 32 (1987) 97-130.

[DB88] T. Dean and M. Boddy, An analysis of time-dependent planning, *Proc. Seventh Nat. Conf. on Artificial Intelligence* (1988) 49-54.

[DE93] T. Dean, L. Kaelbling, J. Kirman and A. Nicholson, Planning with deadlines in stochastic domains, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (1993) 574-579.

[DR91] J. Drakopoulos, FPR: A fuzzy pattern recognizer based on sigmoidals, Technical Report KSL-91-75, Knowledge Systems Laboratory, Stanford University (1991).

[FA90] M. Factor, The process trellis software architecture for parallel real-time monitors, Ph.D. Dissertation, Yale University, December, 1990.

[FA80] L. Fagan, VM: representing time-dependent relations in a medical setting, Ph.D. Dissertation, Computer Science Department, Stanford University, June, 1980.

[FA91] U. Fayyad, *On the induction of decision trees for multiple concept learning*, Ph.D. dissertation, EECS Department, University of Michigan (1991).

[FA92] U. Fayyad, K. Irani, The attribute selection problem in decision tree generation, *Proceedings of the Tenth National Conference on Artificial Intelligence* (1992) 104-110.

[FI72] R. Fikes, P. Hart and N. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence* 3 (1972) 251-288.

[FO84] K. Forbus, Qualitative process theory, *Artificial Intelligence* 24 (1984) 85-168.

[GA86] A. Garvey, M. Hewett, V. Johnson, R. Schulman and B. Hayes-Roth, BB1 user manual -- Common Lisp version, Technical Report KSL-86-61, Knowledge Systems Laboratory, Stanford University (1986).

[GI89] M. Ginsburg, Universal planning: an (almost) universally bad idea, *AI Magazine* 4 (Winter, 1989) 40-44.

[HR85] B. Hayes-Roth, A blackboard architecture for control, *Artificial Intelligence* 26 (1985) 251-321.

[HR92] B. Hayes-Roth, R. Washington, D. Ash, R. Hewett, A. Collinot, A. Vina and A. Seiver, Guardian: a prototype intelligent agent for intensive-care monitoring, *Artificial Intelligence in Medicine* 4 (1992) 165-185.

[HE90] J. Hendler, A. Agrawala, Mission critical planning: AI on the MARUTI real-time operating system, *Proc. Workshop on Innovative Approaches to Planning, Scheduling, and Control*, ed. K. Sycara (1990) 77-84.

[HE91] M. Henrion, Search-based methods to bound diagnostic probabilities in very large belief nets, *Uncertainty and Artificial Intelligence: Proceedings of the Seventh Conference* (1991) 142-150.

[HE89] R. Hewett, B. Hayes-Roth and A. Seiver, Using prime models in model-based reasoning, *Model-Based Reasoning Workshop, Eleventh International Joint Conference on Artificial Intelligence* (1989).

[HO47] P. Hoel, *Introduction to Mathematical Statistics*, New York: John Wiley and Sons (1947).

[HO87] E. Horvitz, Reasoning about beliefs and actions under computational resource constraints, *Proc. 1987 AAAI Workshop on Uncertainty in Artificial Intelligence* (1987).

[KA87] L. Kaelbling, An architecture for intelligent reactive systems, *Reasoning About Actions and Plans*, eds. M. Georgeff and A. Lansky (1987) 395-410.

[KA88] L. Kaelbling, Goals as parallel program specifications, *Proceedings of the Seventh National Conference on Artificial Intelligence* (1988).

[KA90] L. Kaelbling, Specifying complex behavior for computer agents, *Proceedings: Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990) 433-438.

[KU86] B. Kuipers, Qualitative simulation, *Artificial Intelligence* 29 (1986) 289-338.

[MU93] S. Murthy, S. Kasif, S. Salzberg and R. Beigel, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (1993) 322-327.

[QU83] R. Quinlan, Inductive inference as a tool for the construction of high-performance programs, *Machine Learning*, eds. R. Michalski, T. Mitchell, and J. Carbonell, Palo Alto, Calif.: Tioga (1983).

[QU90] R. Quinlan, Probabilistic decision trees, in *Machine Learning: An Artificial Intelligence Approach, Volume III*, eds. Y. Kodratoff and R. Michalski, San Mateo, Calif.: Morgan Kaufmann (1990).

[RE87] R. Reiter, A theory of diagnosis from first principles, *Artificial Intelligence* 32 (1987) 57-95.

[RO89] S. Rosenschein, Synthesizing information-tracking automata from environment descriptions, *Proceedings of Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, Calif.: Morgan Kaufmann (1989).

[RU91] S. Russell and E. Wefald, Principles of metareasoning, *Artificial Intelligence* 49 (1991) 361-395.

[RL91] G. Rutledge, Dynamic selection of models under time constraints, *Proceedings of Second Annual Conference on AI Simulation and Planning in High Autonomy Systems*, Cocoa Beach (1991) 60-67.

[RL93] G. Rutledge, G. Thomsen, B. Farr, M. Tovar, J. Polaschek, I. Beinlich, L. Sheiner and L. Fagan, The design and implementation of a ventilator-management advisor, *Artificial Intelligence in Medicine* (1993).

[SC87] M. Schoppers, Universal plans for reactive robots in unpredictable environments, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (1987).

[SH76] E. Shortliffe, *MYCIN: Computer-based consultations in medical therapeutics*, New York: American Elsevier (1976).

[TO91] D. Tong, Weaning patients from mechanical ventilation: a knowledge-based system approach, *Computer Methods and Programs in Biomedicine* 35 (1991) 267-278.

[UC93] S. Uckun, B. Dawant and D. Lindstrom, Model-based reasoning in intensive-care monitoring: the YAQ approach, *Artificial Intelligence in Medicine* (1993).

[WA89] R. Washington and B. Hayes-Roth, Input data management in real-time AI systems, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (1989) 250-255.

[WA90] R. Washington and B. Hayes-Roth, Abstraction planning in real-time, Technical Report KSL-90-15, Knowledge Systems Laboratory, Stanford University (1990).

[WI91] L. Widman, A model-based approach to the diagnosis of the cardiac arrhythmias, *Artificial Intelligence in Medicine* 4 (1991) 1-19.

[XU92] J. Xu, S. Hyman and P. King, Knowledge-based flash evoked potential recognition system, *Artificial Intelligence in Medicine* 4(2) (1992) 93-109.